Solution of
Knative Praktikum

Author: Richard Clauß
Docent: Prof. Dr. Martin Leischner
Date: 13.12.2022

Hochschule
Bonn-Rhein-Sieg

Agenda

Task 1 - Definition of serverless computing

Task 2 - Install knative on your kubernetes cluster

Task 3 - Cold-start-delay

Task 4 - Revisions and Traffic Splitting

Task 5 - Autoscaling

Task 6 - Domains

Task 7 - Knative Eventing

Task 1

Definition of serverless computing

1.1 Your attempt to find the main properties of serverless computing

Which properties are defining serverless computing?

Would you still call it serverless computing without property xyz?

A) Functions must run in **containers**
B) Must do **automatic horizontal scaling**
C) Must **scale transparently** to the programmer
D) **No upper limit for scaling out**
E) Must include **load balancing** to function instances
F) Must be able to **scale-to-zero**
G) **On-demand billing**
H) Very fine granular **billing based on milliseconds or seconds execution time**
I) **Fast starting function instances**
J) Must **run in private or public cloud**
K) Must run on **kubernetes**
L) **No sessions** between client and serverless application allowed (for example realized with cookies)
M) **Always cheaper** than other hosting technologies
N) Function **execution time is limited**
O) Must accept **http or https**

1.1 Your attempt to find the main properties of serverless computing

Which properties are defining serverless computing?
➔ No perfect answers possible, because the definition is derived from practise.
It depends on the respective author if features like scale-to-zero are seen as mandatory.
Function-as-a-service can be seen as special case of serverless computing with additional mandatory properties.

Only my opinion and no scientific consensus:

A) Functions must run in **containers** **➔ No (although most if not all solutions use OCI containers)**

B) Must do **automatic horizontal scaling** **➔ Yes**

C) Must **scale transparently** to the programmer **➔ Yes**

D) **No upper limit for scaling out ➔ No (an upper limit is common for example to limit cost in AWS lambda)**

E) Must include **load balancing** to function instances **➔ Yes**

F) Must be able to **scale-to-zero ➔ No**

(very debatable, in my opinion if scale-to-zero is disabled in Knative or AWS Lambda, both can still be called serverless computing, also thinkable: Yes for FaaS in public clouds and No for private clouds)

G) **On-demand billing ➔ No for private clouds (usually dedicated resources), Yes (for FaaS in public clouds)**

H) Very fine granular **billing based on milliseconds or seconds execution time ➔ In general No, but Yes for FaaS**

I) **Fast starting function instances ➔ No**

(common requirement, but no must have property if scale-to-zero is also optional and cold-starts are fully avoidable)

J) Must **run in private or public cloud ➔ No (for example “fn project” runs on any virtual machine)**

K) Must run on **kubernetes ➔ No**

L) **No sessions** between client and serverless application allowed (for example realized with cookies)
➔ No (externalizing state makes a context possible)

M) **Always cheaper** than other hosting technologies **➔ No (depends on traffic shape and pricing)**

N) Function **execution time is limited ➔ No**

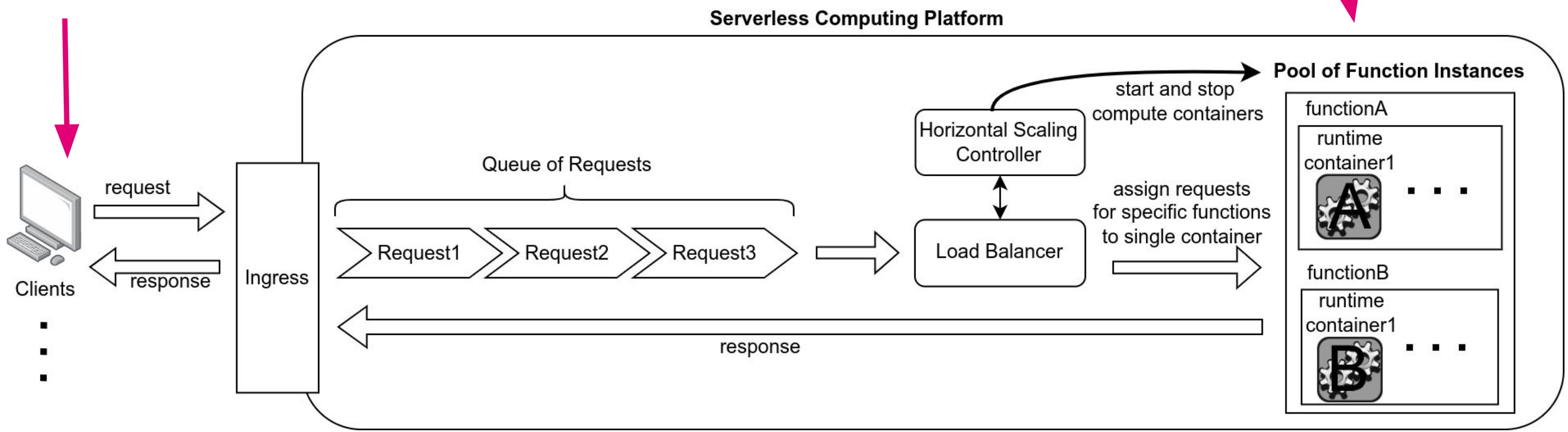
O) Must accept **http or https ➔ No (no must have property, although common)**

Sources: Please refer to [1] in Serverless VL

1.2 My attempt to find the main properties of serverless computing

Scales transparently, horizontally, automatically and often down-to-zero

Different event sources possible


Serverless Computing Platform
Clients
request
response
Ingress
Queue of Requests
Request1
Request2
Request3
Horizontal Scaling Controller
Load Balancer
start and stop compute containers
assign requests for specific functions to single container
response
Pool of Function Instances
functionA
runtime container1
A
functionB
runtime container1
B

Broadest thinkable serverless computing definition:
“Horizontally, transparently and automatically scaling programs are executed in response to events”

Task 2

Install knative on your kubernetes cluster

2.1 Install Knative

See Knative Practical Introduction

Task 3

Cold-start Delay

3.1 Task 3 - Cold-start-delay

a) Describe the cases in which a cold-start-delay can occur

Cold-start

- Before a function invocation can be processed, a new function instance needs to be started
  → cold-start delay = initialization time + execution time

Warm-start

- The invocation can be forwarded to an already existing function instance with free capacity
  → warm-start delay = execution time

Occurrence of cold-starts

- a) When no function instances are running
  - after scale-to-zero, failure or deployment
- b) During scaling out if all present function instances can’t serve more requests.
  - various scaling algorithm dependent reasons possible

3.1 Task 3 - Cold-start-delay

b) Create a service in knative that uses the standard knative pod autoscaler and for which the occurrence of cold-start-delays is impossible

# Case 1: Avoid cold-start after scale-to-zero:

The value enable-scale-to-zero can be set to "false" globally in autoscaler.yaml, but this setting can’t be applied to single revisions.

```
  # Scale to zero feature flag.
  enable-scale-to-zero: "true"
```

See [1,2]

Setting scale-to-zero globally is most likely no best practise

3.1 Task 3 - Cold-start-delay

b) Create a service in knative that uses the standard knative pod autoscaler and for which the occurrence of cold-start-delays is impossible

# Case 1: Avoid cold-start after scale-to-zero:

Alternatively scale-to-zero can be disabled per revision by setting a lower scale bound >=1

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: helloworld-go
  namespace: default
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/min-scale: "1"
    spec:
      containers:
        - image: gcr.io/knative-samples/helloworld-go
```

See [3]

3.1 Task 3 - Cold-start-delay

b) Create a service in knative that uses the standard knative pod autoscaler and for which the occurrence of cold-start-delays is impossible

# Case 2: Avoid cold-start after failure or deployment:

The default initial-scale scale of 1 in the configmap autoscaler.yaml ensures the creation of at least one pod after failure or deployment without further modifications.

```
  # initial-scale is the cluster-wide default value for the initial target
  # scale of a revision after creation, unless overridden by the
  # "autoscaling.knative.dev/initialScale" annotation.
  # This value must be greater than 0 unless allow-zero-initial-scale is true.
    initial-scale: "1"
```

(Per-revision annotation key: autoscaling.knative.dev/initial-scale)

Setting a min-scale of 1 can also achieve the same functionality.

See [1]

3.1 Task 3 - Cold-start-delay

b) Create a service in knative that uses the standard knative pod autoscaler and for which the occurrence of cold-start-delays is impossible

# Case 3: Avoid cold-start through hard-limits during scaling-out

First we verify that a soft and no hard concurrency-limit is applied

Excerpt of configmap defaults.yaml:

```
  # container-concurrency specifies the maximum number
  # of requests the Container can handle at once, and requests
  # above this threshold are queued.  Setting a value of zero
  # disables this throttling and lets through as many requests as
  # the pod receives.
  container-concurrency: "0"
```

-> no hard-limit is the default, there is no change necessary

See: [4,5]

3.1 Task 3 - Cold-start-delay

b) Create a service in knative that uses the standard knative pod autoscaler and for which the occurrence of cold-start-delays is impossible

# Case 4: Avoid cold-start through exceeding target-burst-capacity during scaling-out

If a traffic burst is too large for the application to handle (greater concurrency than target-burst-capacity), the Activator will buffer requests until the capacity could be increased.

We need to set the target-burst-capacity to 0, which means "the Activator is only in path when scaled to 0".

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  annotations:
  name: <service_name>
  namespace: default
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/target-burst-capacity: "0"
```

See [1,6]

3.1 Task 3 - Cold-start-delay

b) Create a service in knative that uses the standard knative pod autoscaler and for which the occurrence of cold-start-delays is impossible

Result:

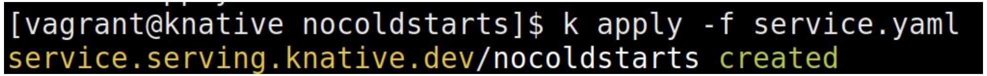
No global changes were made

Resulting service.yaml:

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: nocoldstarts
  namespace: default
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/min-scale: "1"
        autoscaling.knative.dev/target-burst-capacity: "0"
    spec:
      containers:
      - image: gcr.io/knative-samples/helloworld-go
        env:
        - name: TARGET
          value: "nocoldstarts"
        imagePullPolicy: Never
```

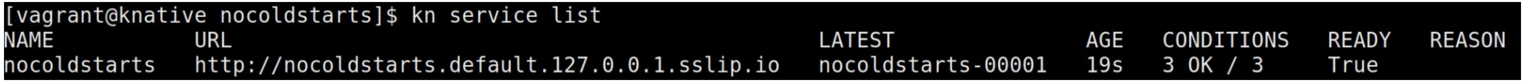
3.1 Task 3 - Cold-start-delay

```
## a) set up and show service nocoldstarts

$ k apply -f service.yaml
```



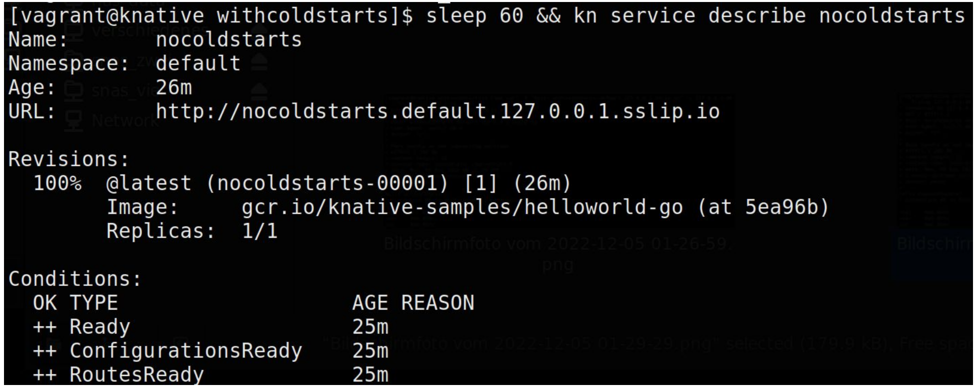
```
[vagrant@knative nocoldstarts]$ k apply -f service.yaml
service.serving.knative.dev/nocoldstarts created
```

```
$ kn service list
```



```
[vagrant@knative nocoldstarts]$ kn service list
NAME           URL                                              LATEST                 AGE   CONDITIONS   READY   REASON
nocoldstarts   http://nocoldstarts.default.127.0.0.1.sslip.io   nocoldstarts-00001     19s   3 OK / 3     True
```

```
$ sleep 60 && kn service describe nocoldstarts
```



```
[vagrant@knative withcoldstarts]$ sleep 60 && kn service describe nocoldstarts
Name:       nocoldstarts
Namespace:  default
Age:        26m
URL:        http://nocoldstarts.default.127.0.0.1.sslip.io

Revisions:
  100%  @latest (nocoldstarts-00001) [1] (26m)
        Image:     gcr.io/knative-samples/helloworld-go (at 5ea96b)
        Replicas:  1/1

Conditions:
  OK TYPE                   AGE REASON
  ++ Ready                  25m
  ++ ConfigurationsReady    25m
  ++ RoutesReady            25m
```

3.1 Task 3 - Cold-start-delay

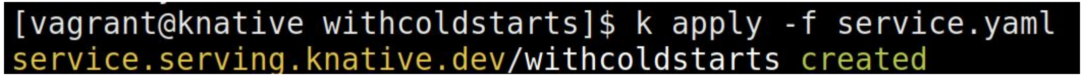
c) Wait one minute to simulate that the service didn't receive any traffic for one minute. Verify that no cold start occurs when sending a request to the service.

Set up an identical service, except that it is configured to perform scale-to-zero. This way a typical cold-start delay can be measured:

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: withcoldstarts
  namespace: default
spec:
  template:
    #metadata:
    #  annotations:
    #    autoscaling.knative.dev/min-scale: "1"
    #    autoscaling.knative.dev/target-burst-capacity: "0"
    spec:
      containers:
      - image: gcr.io/knative-samples/helloworld-go
        env:
        - name: TARGET
          value: "withcoldstarts"
        imagePullPolicy: Never
```

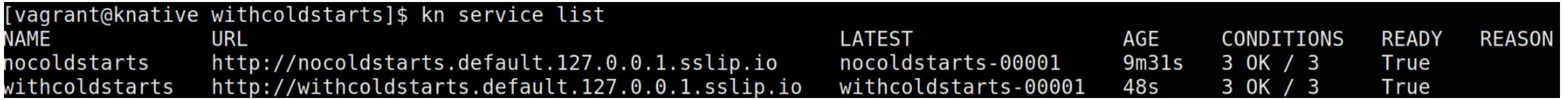
3.1 Task 3 - Cold-start-delay

```
## a) set up and show service withcoldstarts

$ k apply -f service.yaml
```



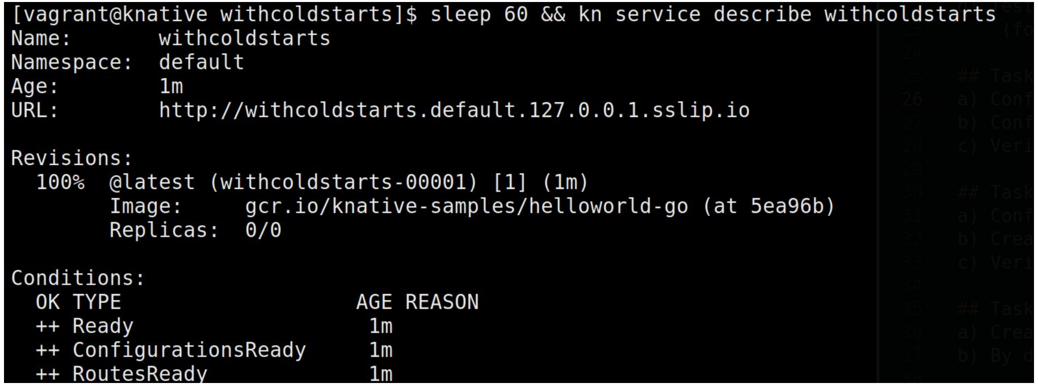
```
[vagrant@knative withcoldstarts]$ k apply -f service.yaml
service.serving.knative.dev/withcoldstarts created
```

```
$ kn service list
```



```
[vagrant@knative withcoldstarts]$ kn service list
NAME             URL                                           LATEST                  AGE     CONDITIONS   READY   REASON
nocoldstarts     http://nocoldstarts.default.127.0.0.1.sslip.io     nocoldstarts-00001      9m31s   3 OK / 3     True
withcoldstarts   http://withcoldstarts.default.127.0.0.1.sslip.io   withcoldstarts-00001    48s     3 OK / 3     True
```

```
$ sleep 60 && kn service describe withcoldstarts
```

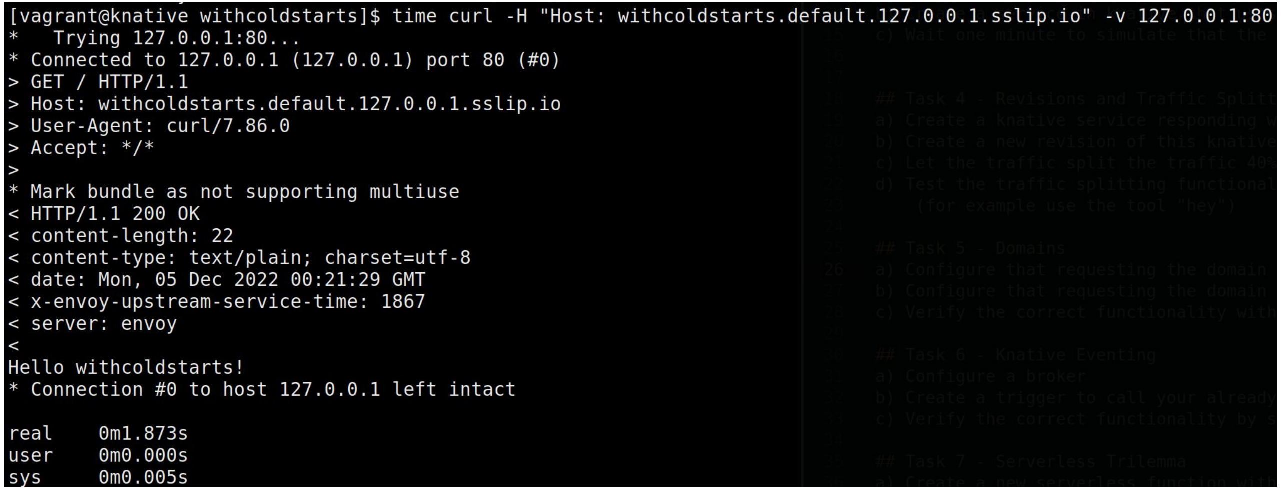


```
[vagrant@knative withcoldstarts]$ sleep 60 && kn service describe withcoldstarts
Name:         withcoldstarts
Namespace:    default
Age:          1m
URL:          http://withcoldstarts.default.127.0.0.1.sslip.io

Revisions:
  100%  @latest (withcoldstarts-00001) [1] (1m)
        Image:     gcr.io/knative-samples/helloworld-go (at 5ea96b)
        Replicas:  0/0

Conditions:
  OK TYPE                   AGE REASON
  ++ Ready                   1m
  ++ ConfigurationsReady     1m
  ++ RoutesReady             1m
```

```
 -> no replicas, scale-to-zero happened
```

3.1 Task 3 - Cold-start-delay

```
## b) compare the delays of both services

$ time curl -H "Host: withcoldstarts.default.127.0.0.1.sslip.io" -v 127.0.0.1:80
```

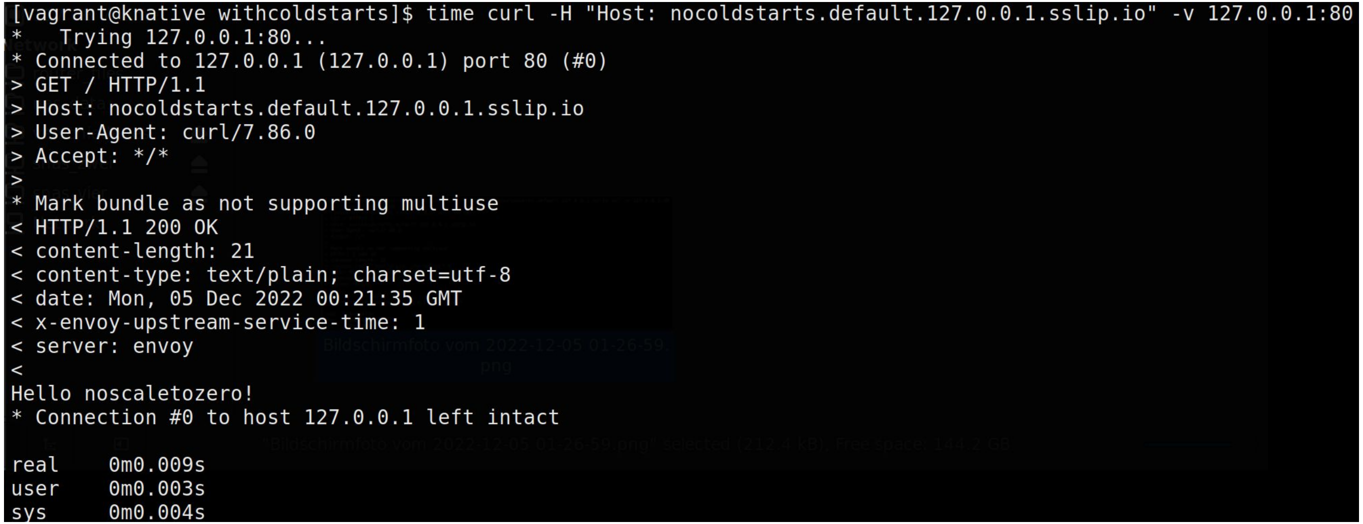


```
[vagrant@knative withcoldstarts]$ time curl -H "Host: withcoldstarts.default.127.0.0.1.sslip.io" -v 127.0.0.1:80
*   Trying 127.0.0.1:80...
* Connected to 127.0.0.1 (127.0.0.1) port 80 (#0)
> GET / HTTP/1.1
> Host: withcoldstarts.default.127.0.0.1.sslip.io
> User-Agent: curl/7.86.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< content-length: 22
< content-type: text/plain; charset=utf-8
< date: Mon, 05 Dec 2022 00:21:29 GMT
< x-envoy-upstream-service-time: 1867
< server: envoy
<
Hello withcoldstarts!
* Connection #0 to host 127.0.0.1 left intact

real    0m1.873s
user    0m0.000s
sys     0m0.005s
```

```
withcoldstarts -> cold-start delay = 0m1.873s
```

3.1 Task 3 - Cold-start-delay

```
## b) compare the delays of both services

$ time curl -H "Host: nocoldstarts.default.127.0.0.1.sslip.io" -v 127.0.0.1:80
```



```
[vagrant@knative withcoldstarts]$ time curl -H "Host: nocoldstarts.default.127.0.0.1.sslip.io" -v 127.0.0.1:80
*   Trying 127.0.0.1:80...
* Connected to 127.0.0.1 (127.0.0.1) port 80 (#0)
> GET / HTTP/1.1
> Host: nocoldstarts.default.127.0.0.1.sslip.io
> User-Agent: curl/7.86.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< content-length: 21
< content-type: text/plain; charset=utf-8
< date: Mon, 05 Dec 2022 00:21:35 GMT
< x-envoy-upstream-service-time: 1
< server: envoy
<
Hello noscaletozero!
* Connection #0 to host 127.0.0.1 left intact

real    0m0.009s
user    0m0.003s
sys     0m0.004s
```

```
withcoldstarts -> cold-start delay = 0m1.873s
nocoldstarts   ->            delay = 0m0.009s

Conclusion: The measured delay of the service nocoldstarts is significantly shorter than a cold-start.
            It is concluded that we see a typical warm-start.
            The conclusion is supported by the fact that we at least see one replica at any point in time.
```

Task 4

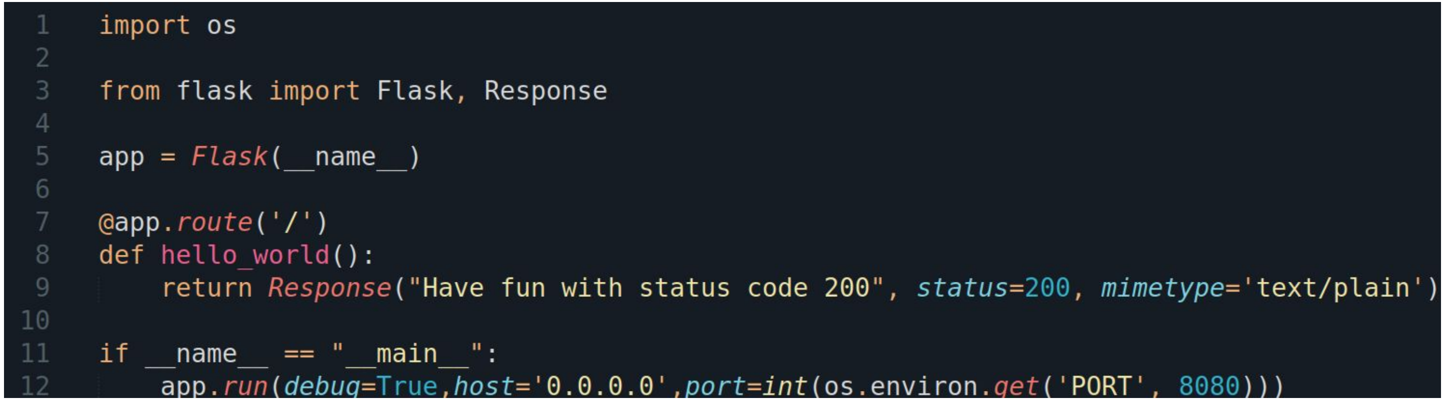
Revisions and Traffic Splitting

4.1 Task 4 - Revisions and Traffic Splitting

a) Create a knative service responding with http status code 200 to all http GET requests

## Use the simple python service from chapter 4 of the introduction as basis and modify it according to our needs:

splitter_v1.0/app.py:



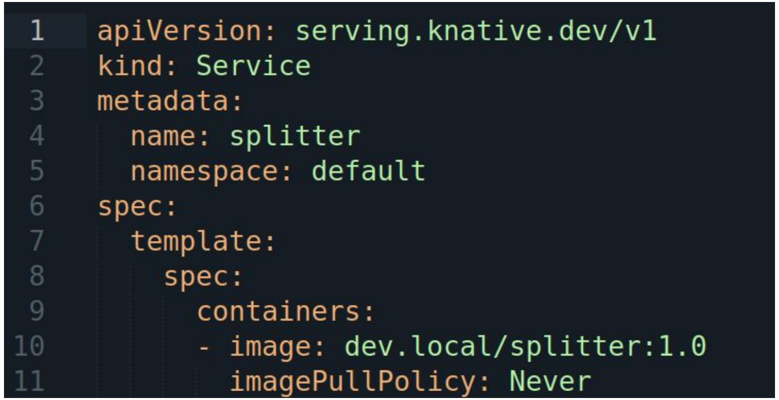
```python
import os

from flask import Flask, Response

app = Flask(__name__)

@app.route('/')
def hello_world():
    return Response("Have fun with status code 200", status=200, mimetype='text/plain')

if __name__ == "__main__":
    app.run(debug=True,host='0.0.0.0',port=int(os.environ.get('PORT', 8080)))
```

4.1 Task 4 - Revisions and Traffic Splitting

a) Create a knative service responding with http status code 200 to all http GET requests

## Use the simple python service from chapter 4 of the introduction as basis and modify it according to our needs:

splitter_v1.0/service.yaml:



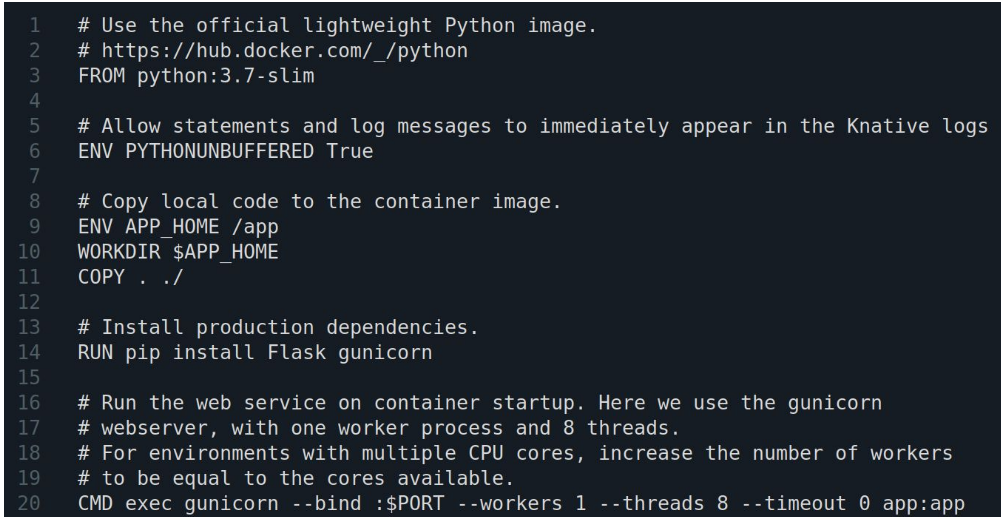
```yaml
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: splitter
  namespace: default
spec:
  template:
    spec:
      containers:
      - image: dev.local/splitter:1.0
        imagePullPolicy: Never
```

4.1 Task 4 - Revisions and Traffic Splitting

a) Create a knative service responding with http status code 200 to all http GET requests

## Use the simple python service from chapter 4 of the introduction as basis and modify it according to our needs:

splitter_v1.0/Dockerfile:



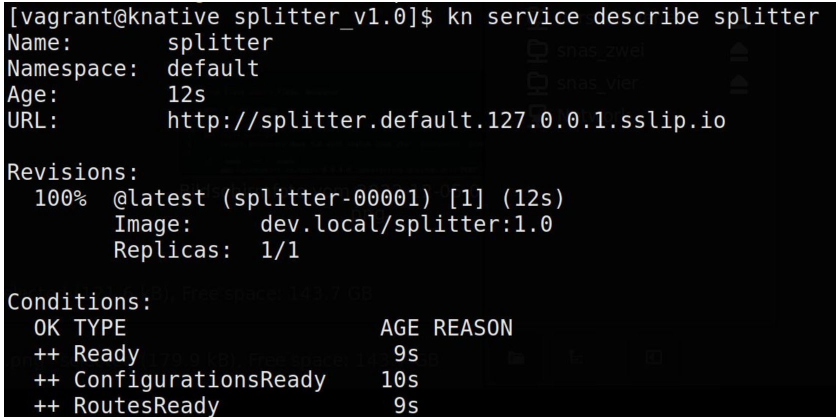
```
# Use the official lightweight Python image.
# https://hub.docker.com/_/python
FROM python:3.7-slim

# Allow statements and log messages to immediately appear in the Knative logs
ENV PYTHONUNBUFFERED True

# Copy local code to the container image.
ENV APP_HOME /app
WORKDIR $APP_HOME
COPY . ./

# Install production dependencies.
RUN pip install Flask gunicorn

# Run the web service on container startup. Here we use the gunicorn
# webserver, with one worker process and 8 threads.
# For environments with multiple CPU cores, increase the number of workers
# to be equal to the cores available.
CMD exec gunicorn --bind :$PORT --workers 1 --threads 8 --timeout 0 app:app
```

4.1 Task 4 - Revisions and Traffic Splitting

a) Create a knative service responding with http status code 200 to all http GET requests

Load the image into kind and create the service:

```
$ cd splitter_v1.0
$ docker build -t splitter:1.0 .
$ docker tag splitter:1.0 dev.local/splitter:1.0
$ kind load docker-image dev.local/splitter:1.0 -n knative
$ k apply -f service.yaml
$ kn service describe splitter
```

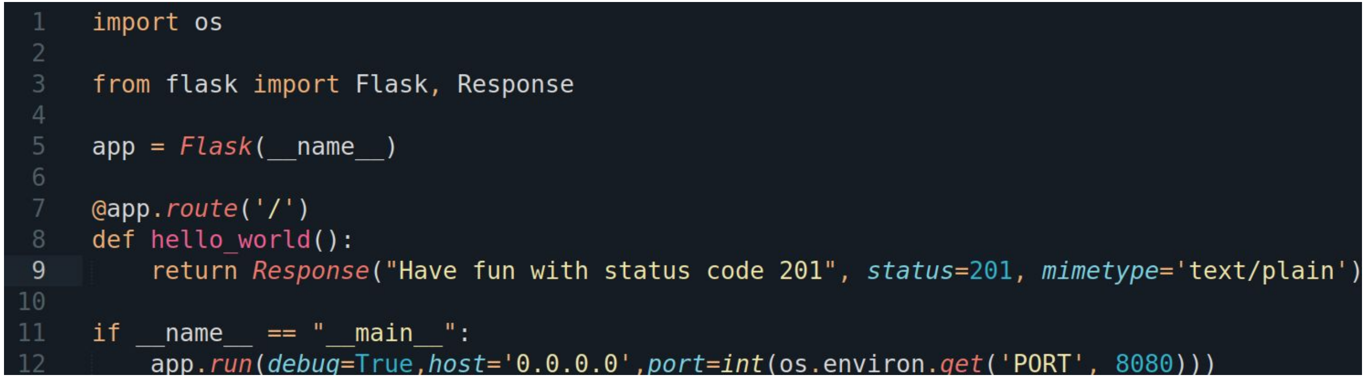


```
[vagrant@knative splitter_v1.0]$ kn service describe splitter
Name:        splitter
Namespace:   default
Age:         12s
URL:         http://splitter.default.127.0.0.1.sslip.io

Revisions:
  100%  @latest (splitter-00001) [1] (12s)
        Image:     dev.local/splitter:1.0
        Replicas:  1/1

Conditions:
  OK TYPE                  AGE REASON
  ++ Ready                  9s
  ++ ConfigurationsReady   10s
  ++ RoutesReady            9s
```

4.1 Task 4 - Revisions and Traffic Splitting

b) Create a new revision of this knative service, which responds with http status code 201 to all http GET requests

splitter_v1.1/app.py:



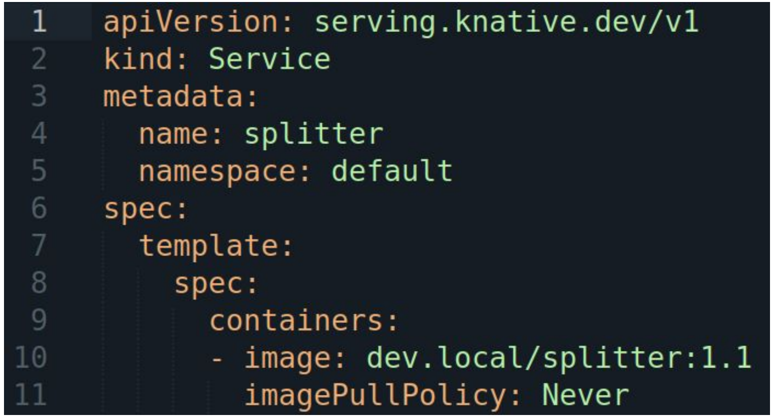
```
import os

from flask import Flask, Response

app = Flask(__name__)

@app.route('/')
def hello_world():
    return Response("Have fun with status code 201", status=201, mimetype='text/plain')

if __name__ == "__main__":
    app.run(debug=True,host='0.0.0.0',port=int(os.environ.get('PORT', 8080)))
```

Sources: [7]

4.1 Task 4 - Revisions and Traffic Splitting

b) Create a new revision of this knative service, which responds with http status code 201 to all http GET requests

splitter_v1.1/service.yaml:



```yaml
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: splitter
  namespace: default
spec:
  template:
    spec:
      containers:
      - image: dev.local/splitter:1.1
        imagePullPolicy: Never
```

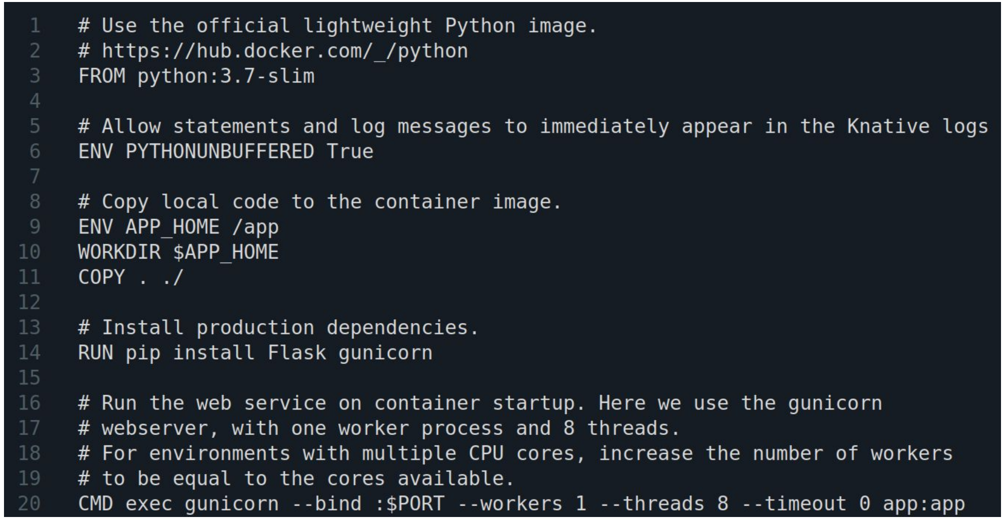
(Because the service name is the same we can have 2 folders with different files changing the same knative service)

Sources: [7]

4.1 Task 4 - Revisions and Traffic Splitting

b) Create a new revision of this knative service, which responds with http status code 201 to all http GET requests

splitter_v1.1/Dockerfile:



```
# Use the official lightweight Python image.
# https://hub.docker.com/_/python
FROM python:3.7-slim

# Allow statements and log messages to immediately appear in the Knative logs
ENV PYTHONUNBUFFERED True

# Copy local code to the container image.
ENV APP_HOME /app
WORKDIR $APP_HOME
COPY . ./

# Install production dependencies.
RUN pip install Flask gunicorn

# Run the web service on container startup. Here we use the gunicorn
# webserver, with one worker process and 8 threads.
# For environments with multiple CPU cores, increase the number of workers
# to be equal to the cores available.
CMD exec gunicorn --bind :$PORT --workers 1 --threads 8 --timeout 0 app:app
```

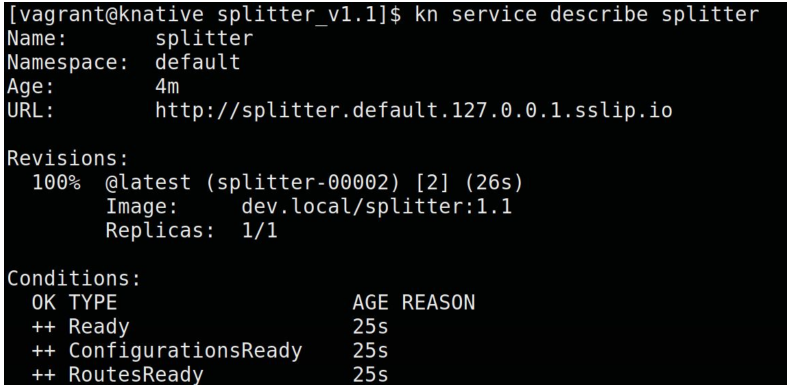
Sources: [7]

4.1 Task 4 - Revisions and Traffic Splitting

b) Create a new revision of this knative service, which responds with http status code 201 to all http GET requests

Load the image into kind and create the service:

```
$ cd splitter_v1.1
$ docker build -t splitter:1.1 .
$ docker tag splitter:1.1 dev.local/splitter:1.1
$ kind load docker-image dev.local/splitter:1.1 -n knative
$ k apply -f service.yaml
$ kn service describe splitter
```



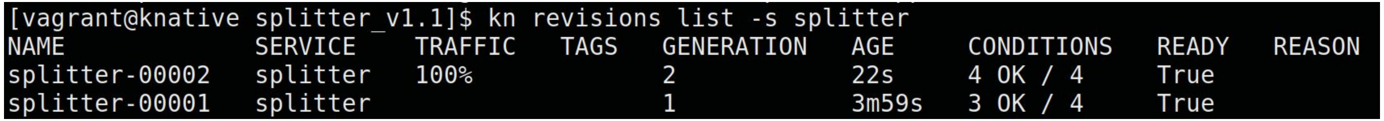
```
[vagrant@knative splitter_v1.1]$ kn service describe splitter
Name:       splitter
Namespace:  default
Age:        4m
URL:        http://splitter.default.127.0.0.1.sslip.io

Revisions:
  100%  @latest (splitter-00002) [2] (26s)
        Image:     dev.local/splitter:1.1
        Replicas:  1/1

Conditions:
  OK TYPE                    AGE REASON
  ++ Ready                   25s
  ++ ConfigurationsReady     25s
  ++ RoutesReady             25s
```

Sources: [7]

4.1 Task 4 - Revisions and Traffic Splitting

b) Create a new revision of this knative service, which responds with http status code 201 to all http GET requests

```
$ kn revisions list -s splitter
```



```
[vagrant@knative splitter_v1.1]$ kn revisions list -s splitter
NAME             SERVICE    TRAFFIC   TAGS   GENERATION   AGE     CONDITIONS   READY   REASON
splitter-00002   splitter   100%             2            22s     4 OK / 4     True
splitter-00001   splitter                    1            3m59s   3 OK / 4     True
```

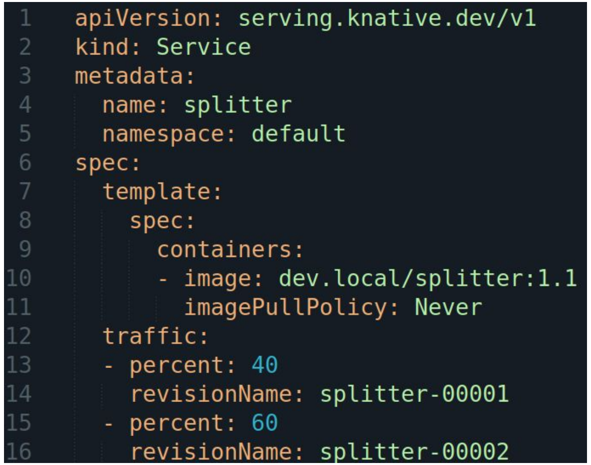
Sources: [7]

4.1 Task 4 - Revisions and Traffic Splitting

c) Let knative split the incoming traffic 40% to the first revision and 60% to the second revision

```
set up traffic splitting by editing the service.yaml or use
$ k edit kservice splitter
```

Result:



```yaml
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: splitter
  namespace: default
spec:
  template:
    spec:
      containers:
      - image: dev.local/splitter:1.1
        imagePullPolicy: Never
  traffic:
  - percent: 40
    revisionName: splitter-00001
  - percent: 60
    revisionName: splitter-00002
```

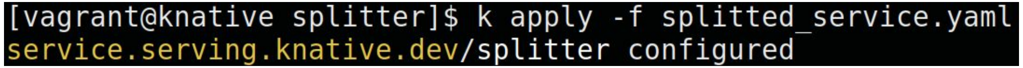
```
$ k apply -f splitted_service.yaml
```

Sources: [7,8]

4.1 Task 4 - Revisions and Traffic Splitting

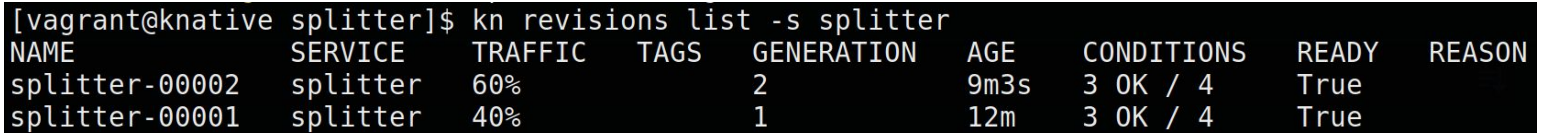
c) Let knative split the incoming traffic 40% to the first revision and 60% to the second revision

```
$ k apply -f splitted_service.yaml
```



```
[vagrant@knative splitter]$ k apply -f splitted_service.yaml
service.serving.knative.dev/splitter configured
```

```
$ kn revisions list -s splitter
```



```
[vagrant@knative splitter]$ kn revisions list -s splitter
NAME             SERVICE    TRAFFIC   TAGS   GENERATION   AGE    CONDITIONS   READY   REASON
splitter-00002   splitter   60%              2            9m3s   3 OK / 4     True
splitter-00001   splitter   40%              1            12m    3 OK / 4     True
```

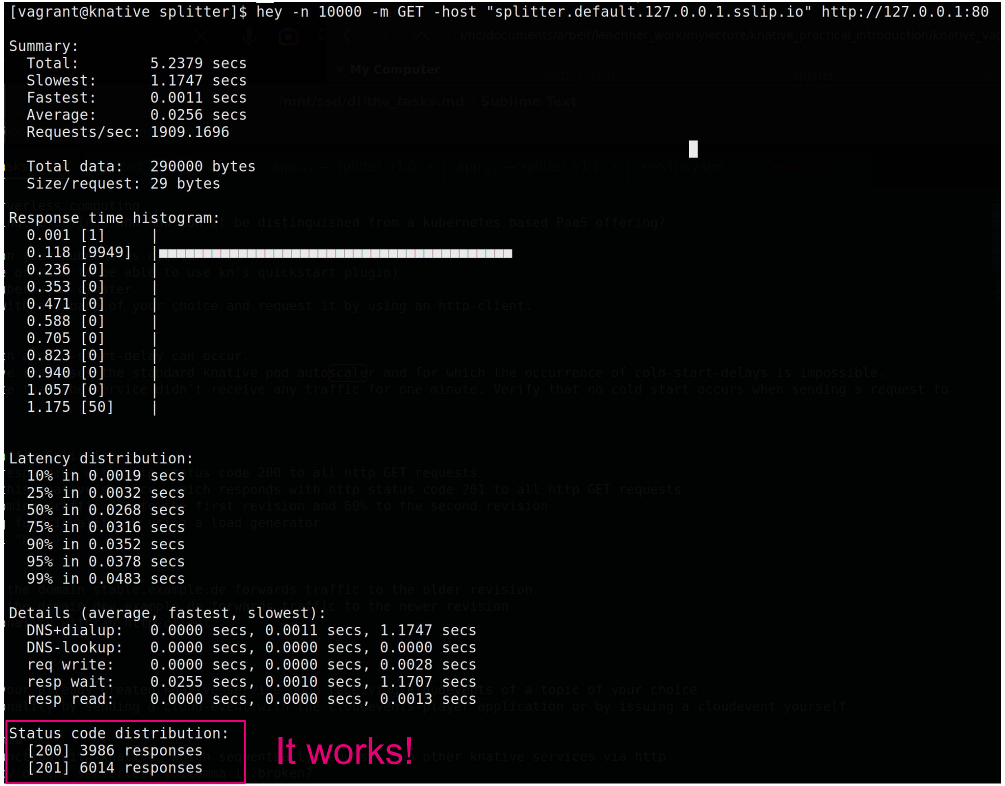
(Note: Changing the traffic distribution doesn’t create a new revision)

Sources: [7,8]

4.1 Task 4 - Revisions and Traffic Splitting

d) Test the traffic splitting functionality by using a load generator

```
$ hey -n 10000 -m GET -host "splitter.default.127.0.0.1.sslip.io" http://127.0.0.1:80
```



```
[vagrant@knative splitter]$ hey -n 10000 -m GET -host "splitter.default.127.0.0.1.sslip.io" http://127.0.0.1:80

Summary:
  Total:        5.2379 secs
  Slowest:      1.1747 secs
  Fastest:      0.0011 secs
  Average:      0.0256 secs
  Requests/sec: 1909.1696

  Total data:   290000 bytes
  Size/request: 29 bytes

Response time histogram:
  0.001 [1]     |
  0.118 [9949]  |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.236 [0]     |
  0.353 [0]     |
  0.471 [0]     |
  0.588 [0]     |
  0.705 [0]     |
  0.823 [0]     |
  0.940 [0]     |
  1.057 [0]     |
  1.175 [50]    |


Latency distribution:
  10% in 0.0019 secs
  25% in 0.0032 secs
  50% in 0.0268 secs
  75% in 0.0316 secs
  90% in 0.0352 secs
  95% in 0.0378 secs
  99% in 0.0483 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.0000 secs, 0.0011 secs, 1.1747 secs
  DNS-lookup:   0.0000 secs, 0.0000 secs, 0.0000 secs
  req write:    0.0000 secs, 0.0000 secs, 0.0028 secs
  resp wait:    0.0255 secs, 0.0010 secs, 1.1707 secs
  resp read:    0.0000 secs, 0.0000 secs, 0.0013 secs

Status code distribution:
  [200] 3986 responses
  [201] 6014 responses
```

It works!

Note: We see cold-start delays for the first 50 requests, because hey sends 50 concurrent requests by default

Task 5

Autoscaling

5.1 Task 5 - Autoscaling

a) Create a knative service with a target concurrency of 10 concurrent requests per replica.
Use the default Knative Pod Autoscaler.

b) Let the new service receive 50 concurrent http-requests from a load generator (for example use the tool "hey").

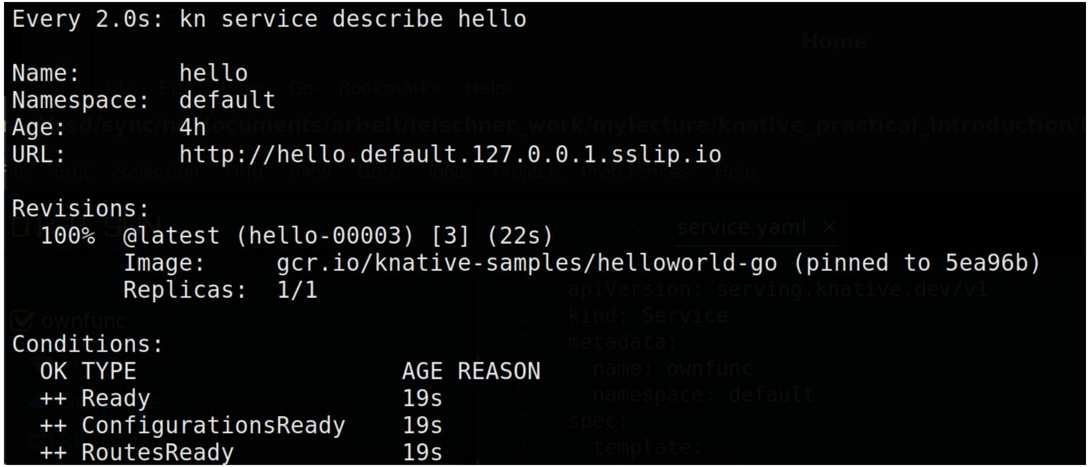
c) You will see that the autoscaling algorithm does not scale the number of replicas to exactly 5 as expected.
How many replicas do you see?

d) Describe the reason for the observed behavior and change the configuration
so that you see exactly 5 replicas for 50 concurrent requests.

Sources: [1,9]

5.1 Task 5 - Autoscaling

```
## a) try to spam hello service (hey sends 50 concurrent requests by default)
$ hey -n 10000000 -m GET -host "hello.default.127.0.0.1.sslip.io" http://127.0.0.1:80

$ watch “kn service describe hello”
```



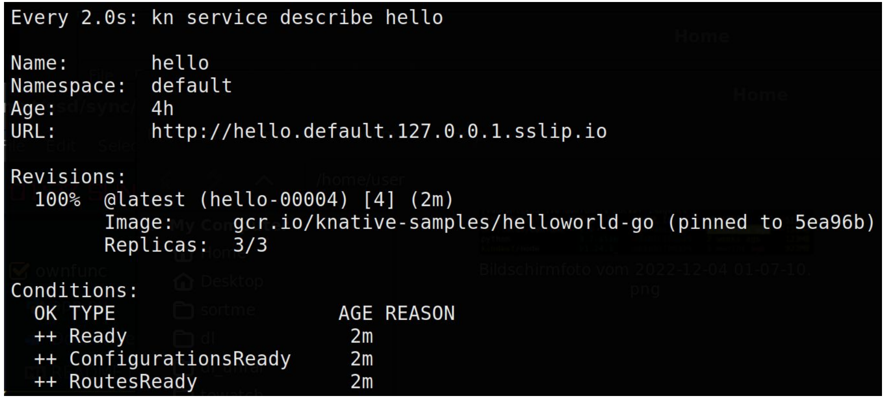
```
Every 2.0s: kn service describe hello

Name:        hello
Namespace:   default
Age:         4h
URL:         http://hello.default.127.0.0.1.sslip.io

Revisions:
  100%  @latest (hello-00003) [3] (22s)
        Image:     gcr.io/knative-samples/helloworld-go (pinned to 5ea96b)
        Replicas:  1/1

Conditions:
  OK TYPE                  AGE REASON
  ++ Ready                 19s
  ++ ConfigurationsReady   19s
  ++ RoutesReady           19s
```

```
No autoscaling although we have 50 concurrent workers sending http-requests?

the ConfigMap autoscaler.yaml gives the answer:
    60s default stable-window                     <- a little bit slow
    container-concurrency-target-default: "100" <- we only send 50 concurrent requests
```

Sources: [1,9]

5.1 Task 5 - Autoscaling

```
## b) fix the concurrency target and window size
$ k edit ksvc hello

spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/stable-window: "1s"
        autoscaling.knative.dev/target: "10"

and try to spam hello service again with 50 concurrent requests
$ hey -n 10000000 -m GET -host "hello.default.127.0.0.1.sslip.io" http://127.0.0.1:80

$ watch “kn service describe hello”
```



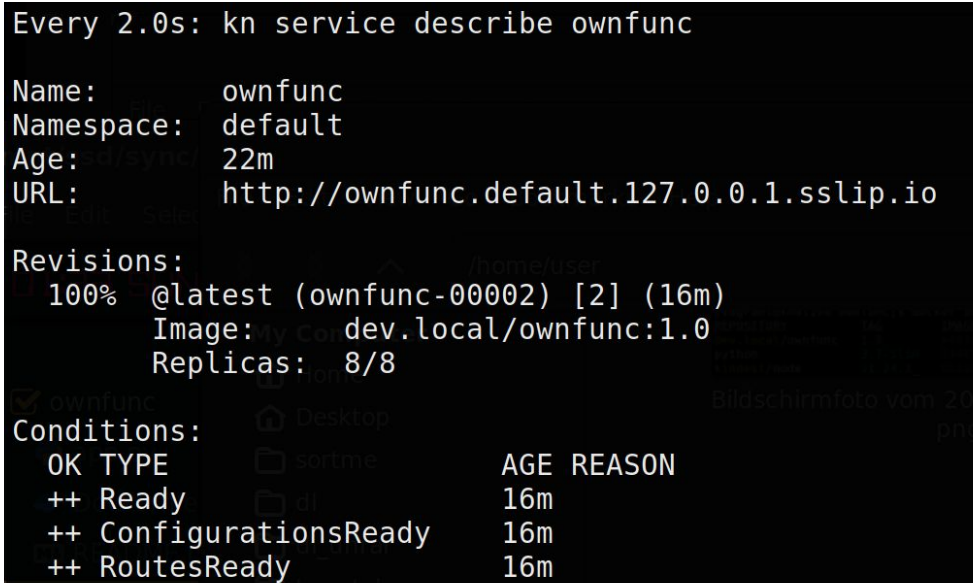
```
Every 2.0s: kn service describe hello

Name:        hello
Namespace:   default
Age:         4h
URL:         http://hello.default.127.0.0.1.sslip.io

Revisions:
  100%  @latest (hello-00004) [4] (2m)
        Image:    gcr.io/knative-samples/helloworld-go (pinned to 5ea96b)
        Replicas:  3/3

Conditions:
  OK TYPE                  AGE REASON
  ++ Ready                  2m
  ++ ConfigurationsReady    2m
  ++ RoutesReady            2m
```

service settles to only 3 pods
→ reason: the pods are too fast
→ The system can’t reach 50 concurrent requests

Sources: [1,9]

5.1 Task 5 - Autoscaling

```
## c) switch from a workload doing “nothing” to the python container
      from chapter 4 of the practical introduction, which sleeps for 1 second on each request.

try to spam ownfunc service with 50 concurrent requests
$ hey -n 10000000 -m GET -host "ownfunc.default.127.0.0.1.sslip.io" http://127.0.0.1:80

$ watch "kn service describe ownfunc"
```



```
Every 2.0s: kn service describe ownfunc

Name:        ownfunc
Namespace:   default
Age:         22m
URL:         http://ownfunc.default.127.0.0.1.sslip.io

Revisions:
  100%  @latest (ownfunc-00002) [2] (16m)
        Image:     dev.local/ownfunc:1.0
        Replicas:  8/8

Conditions:
  OK TYPE                   AGE REASON
  ++ Ready                  16m
  ++ ConfigurationsReady    16m
  ++ RoutesReady            16m
```

why 8? and not 50:10=5 replicas?

→ autoscaler.yaml configmap defines that 70% of the the number of replicas shall provide the target capacity

→ so that scaling up happens earlier than needed

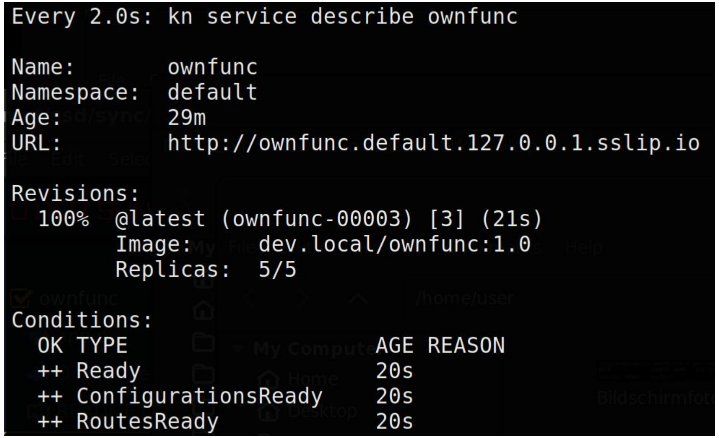
```
container-concurrency-target-percentage: "70"
```

5×(1÷0.7)≈7,14≈8 pods

Sources: [1,9]

5.1 Task 5 - Autoscaling

```
## d) change the target capacity of service ownfunc
$ k edit ksvc ownfunc

spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/target-utilization-percentage: "100"

try to spam ownfunc service again with 50 concurrent requests
$ hey -n 10000000 -m GET -host "ownfunc.default.127.0.0.1.sslip.io" http://127.0.0.1:80

$ watch "kn service describe ownfunc"
```



```
Every 2.0s: kn service describe ownfunc

Name:        ownfunc
Namespace:   default
Age:         29m
URL:         http://ownfunc.default.127.0.0.1.sslip.io

Revisions:
  100%  @latest (ownfunc-00003) [3] (21s)
        Image:     dev.local/ownfunc:1.0
        Replicas:  5/5

Conditions:
  OK TYPE                 AGE REASON
  ++ Ready                20s
  ++ ConfigurationsReady  20s
  ++ RoutesReady          20s
```

Finally exactly 5 replicas as desired!

Sources: [1,9]

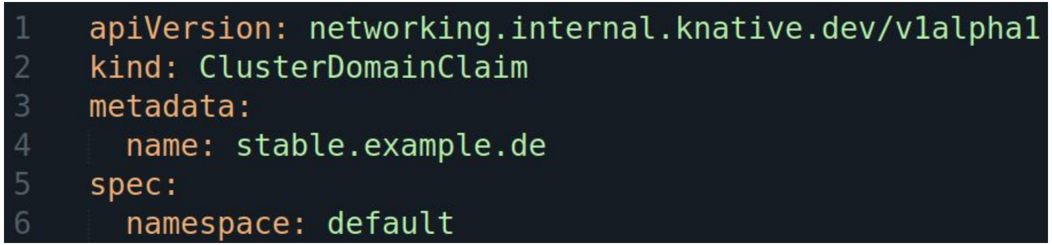
Task 6

Domains

6.1 Task 5 - Domains

a) Configure that requesting the domain stable.example.de via knative's ingress forwards traffic to the service from task 4 (no https needed)

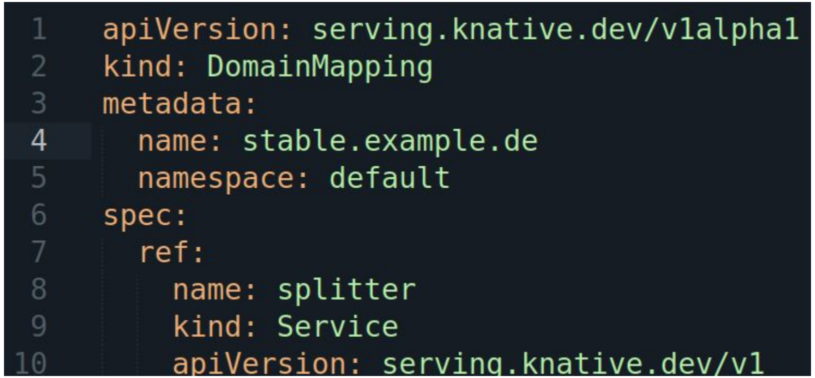
clusterdomainclaim.yaml:



```yaml
apiVersion: networking.internal.knative.dev/v1alpha1
kind: ClusterDomainClaim
metadata:
  name: stable.example.de
spec:
  namespace: default
```

(ClusterDomainClaim delegates the domain name to the namespace you want to create the DomainMapping in)

domainmapping.yaml:



```yaml
apiVersion: serving.knative.dev/v1alpha1
kind: DomainMapping
metadata:
  name: stable.example.de
  namespace: default
spec:
  ref:
    name: splitter
    kind: Service
    apiVersion: serving.knative.dev/v1
```

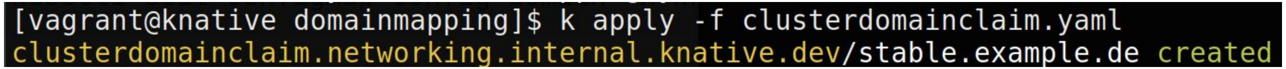
(You can create a DomainMapping object to map a single, non-wildcard domain to a specific Knative Service.)

Sources: [10,11]

6.1 Task 5 - Domains

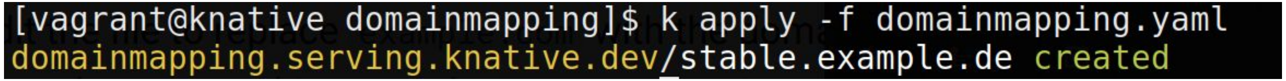
a) Configure that requesting the domain stable.example.de via knative's ingress forwards traffic to the service from task 4 (no https needed)

```
$ k apply -f clusterdomainclaim.yaml
```



```
[vagrant@knative domainmapping]$ k apply -f clusterdomainclaim.yaml
clusterdomainclaim.networking.internal.knative.dev/stable.example.de created
```

```
$ k apply -f domainmapping.yaml
```



```
[vagrant@knative domainmapping]$ k apply -f domainmapping.yaml
domainmapping.serving.knative.dev/stable.example.de created
```

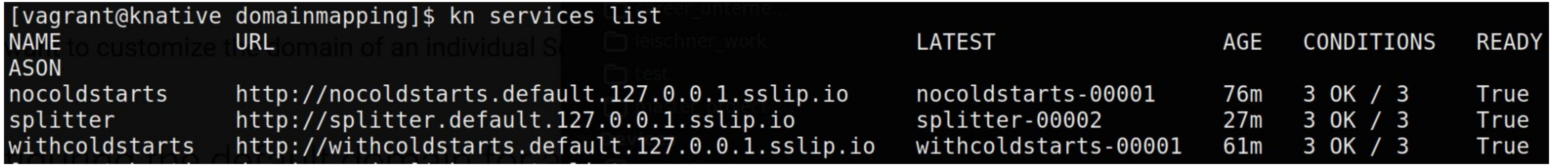
Sources: [10,11]

6.1 Task 5 - Domains

a) Configure that requesting the domain stable.example.de via knative's ingress forwards traffic to the service from task 4 (no https needed)

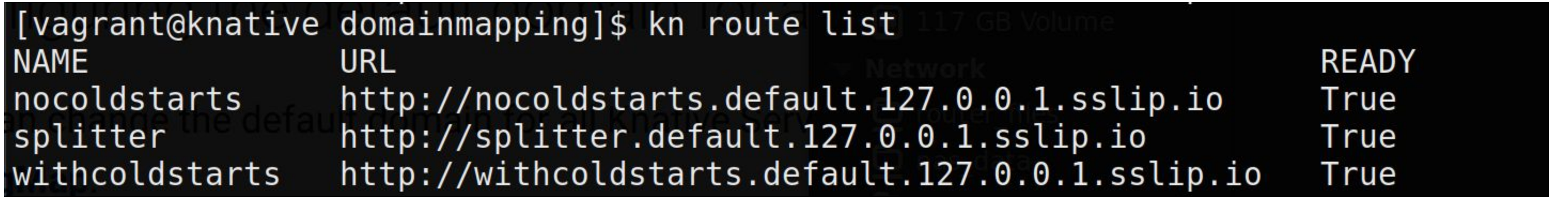
Nothing changed in our services:

$ kn services list



```
[vagrant@knative domainmapping]$ kn services list
NAME            URL                                            LATEST                 AGE   CONDITIONS   READY
ASON
nocoldstarts    http://nocoldstarts.default.127.0.0.1.sslip.io    nocoldstarts-00001     76m   3 OK / 3     True
splitter        http://splitter.default.127.0.0.1.sslip.io        splitter-00002         27m   3 OK / 3     True
withcoldstarts  http://withcoldstarts.default.127.0.0.1.sslip.io  withcoldstarts-00001   61m   3 OK / 3     True
```

Nothing changed in our routes:



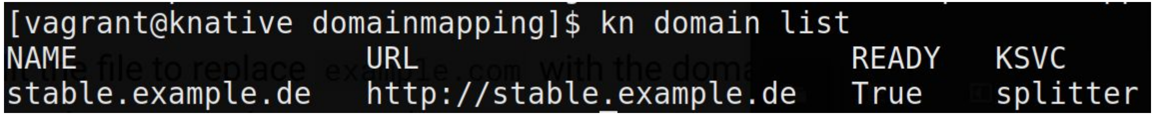
```
[vagrant@knative domainmapping]$ kn route list
NAME            URL                                                READY
nocoldstarts    http://nocoldstarts.default.127.0.0.1.sslip.io     True
splitter        http://splitter.default.127.0.0.1.sslip.io         True
withcoldstarts  http://withcoldstarts.default.127.0.0.1.sslip.io   True
```

Sources: [10,11]

6.1 Task 5 - Domains

a) Configure that requesting the domain stable.example.de via knative's ingress forwards traffic to the service from task 4 (no https needed)

```
The domain management shows the newly created mapping:
$ kn domain list
```



```
[vagrant@knative domainmapping]$ kn domain list
NAME                URL                          READY   KSVC
stable.example.de   http://stable.example.de     True    splitter
```

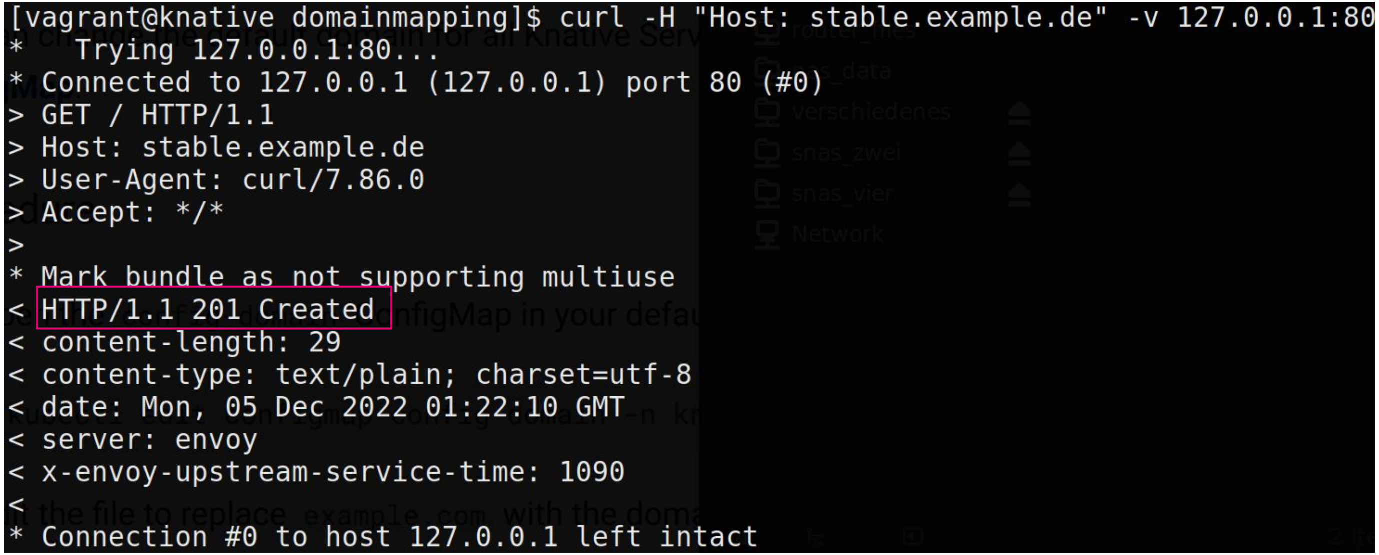
Sources: [10,11]

6.1 Task 5 - Domains

b) Verify the correct functionality of the domain mapping with an http-client

And we can successfully request the domain over the NodePort service leading to knative's kourier ingress controller:

```
$ curl -H "Host: stable.example.de" -v 127.0.0.1:80
```



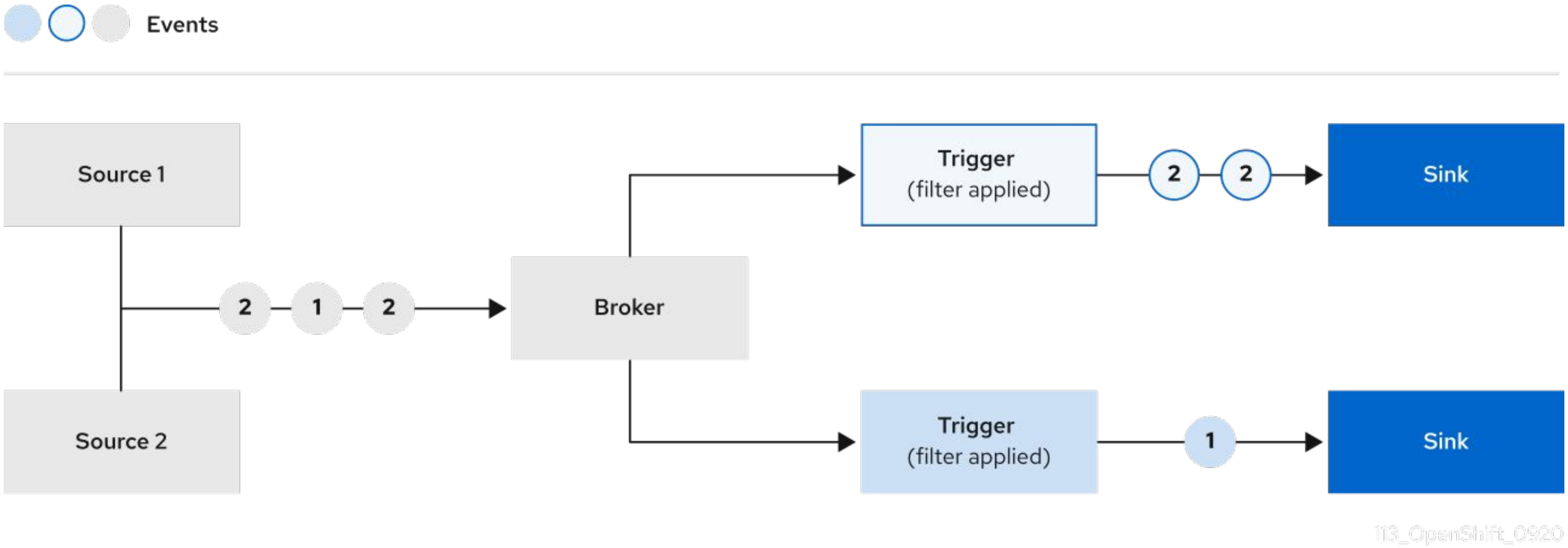
```
[vagrant@knative domainmapping]$ curl -H "Host: stable.example.de" -v 127.0.0.1:80
*   Trying 127.0.0.1:80...
* Connected to 127.0.0.1 (127.0.0.1) port 80 (#0)
> GET / HTTP/1.1
> Host: stable.example.de
> User-Agent: curl/7.86.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 201 Created
< content-length: 29
< content-type: text/plain; charset=utf-8
< date: Mon, 05 Dec 2022 01:22:10 GMT
< server: envoy
< x-envoy-upstream-service-time: 1090
<
* Connection #0 to host 127.0.0.1 left intact
```

Sources: [10,11]

Task 7

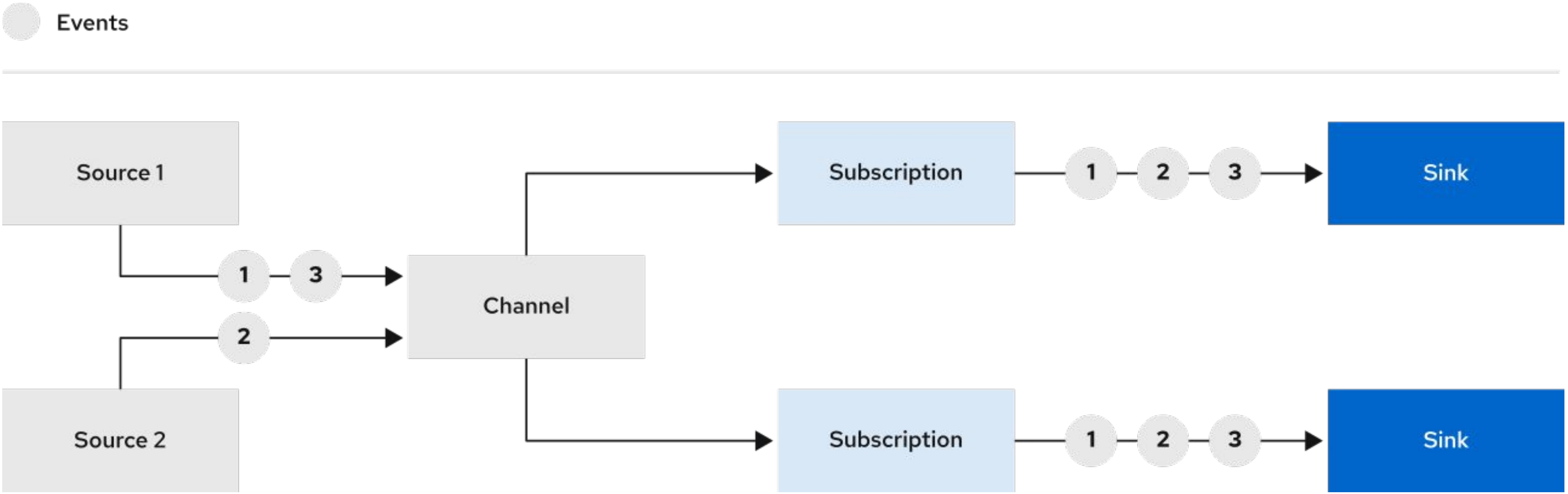
Knative Eventing

7.1 Broker and Trigger



See also: [12,13,14,15]

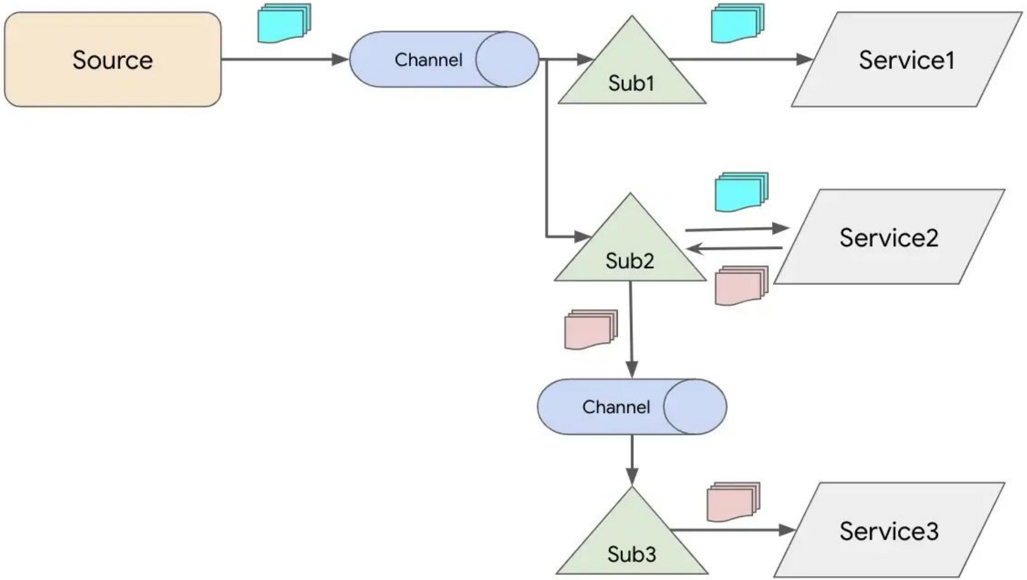
7.2 Channel and Subscriber



See also: [13,14,16,17]

7.3 Complex delivery with channels and subscribers

A) Source sends events over a channel to multiple sinks
Replies of Service2 are forwarded over another channel to a different sink



Sources: [13]

7.4 Complex delivery with channels and subscribers

Example channel:

```
apiVersion: messaging.knative.dev/v1
kind: Channel
metadata:
  name: <example-channel>
  namespace: <namespace>
```

Example subscription:

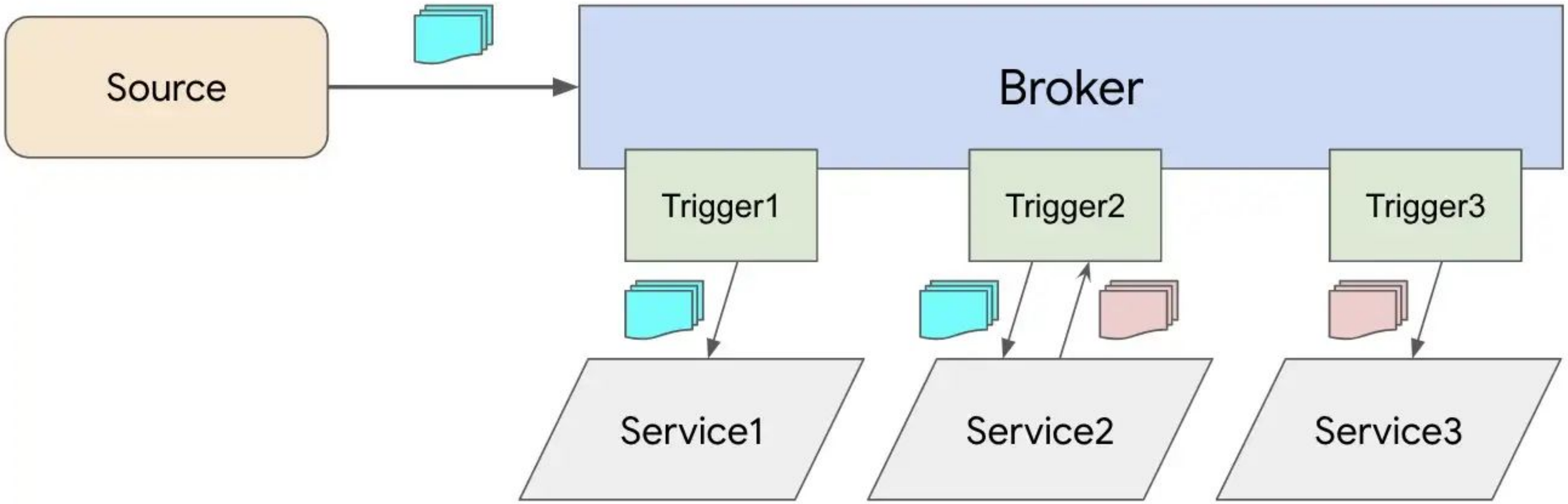
```
kn subscription create <subscription-name> \
  --channel <Group:Version:Kind>:<channel-name> \
  --sink <sink-prefix>:<sink-name> \
  --sink-reply <sink-prefix>:<sink-name> \
  --sink-dead-letter <sink-prefix>:<sink-name>
```

No filters, only pipe plumbing from channel to sink

Sources: [17]

7.5 Broker and Trigger

B) Source sends events to broker
Triggers filter events and forward matching ones to the services
Replies are sent back to the broker



Sources: [13]

7.6 Main Resources of Knative Eventing

Summary:

Channel + Subscriber

- Pipes forwarding all events from one place to another
- Replies can be forwarded to another sink

Broker + Trigger

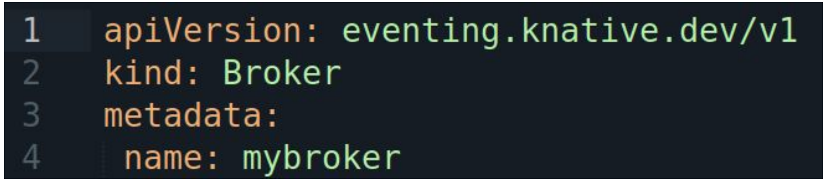
- Broker is central point receiving all events
- Replies are automatically sent back to the broker
- Triggers filter events individually according to their configuration

Sources: [12,13,14,15,16,17]

7.7 Task 6 - Knative Eventing

a) Create a new knative eventing broker, which has the type "Multi-tenant channel-based broker"

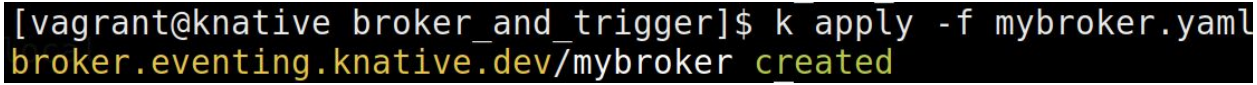
mybroker.yaml:



```
apiVersion: eventing.knative.dev/v1
kind: Broker
metadata:
  name: mybroker
```

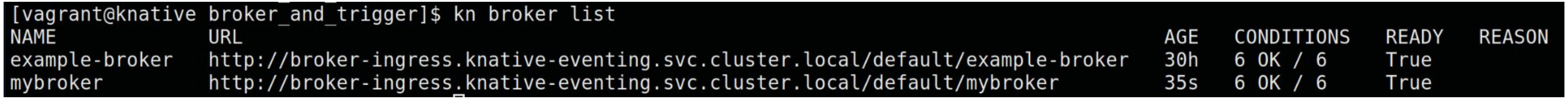
(the desired type is the default type)

$ k apply -f mybroker.yaml



```
[vagrant@knative broker_and_trigger]$ k apply -f mybroker.yaml
broker.eventing.knative.dev/mybroker created
```

$ kn broker list



```
[vagrant@knative broker_and_trigger]$ kn broker list
NAME             URL                                                                     AGE    CONDITIONS   READY   REASON
example-broker   http://broker-ingress.knative-eventing.svc.cluster.local/default/example-broker   30h    6 OK / 6     True
mybroker         http://broker-ingress.knative-eventing.svc.cluster.local/default/mybroker         35s    6 OK / 6     True
```

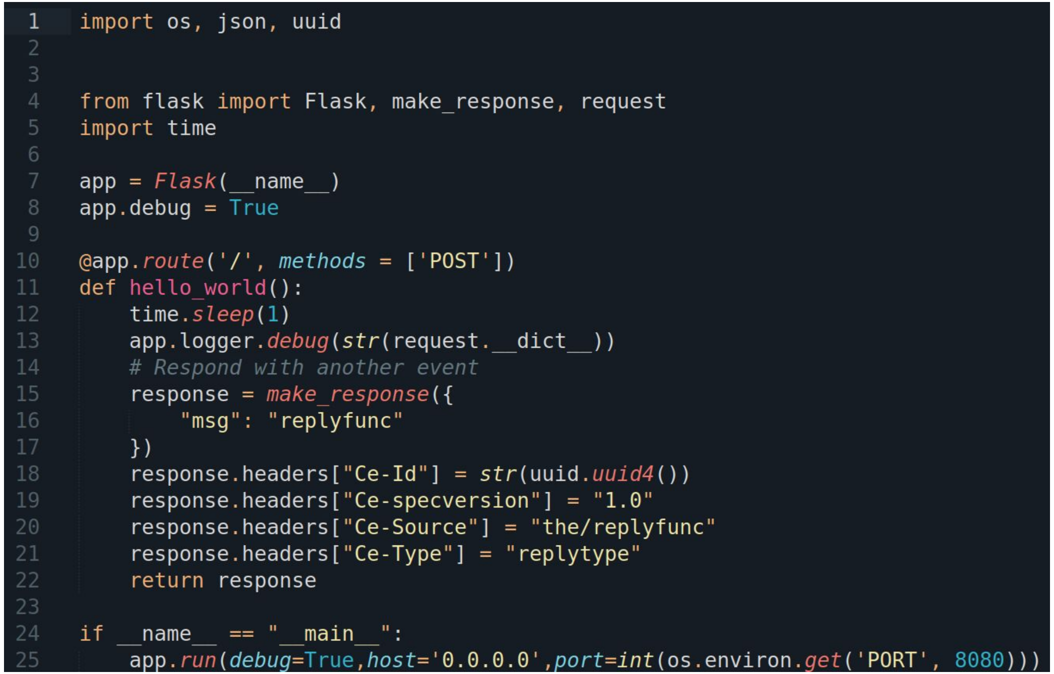
(We can now post CloudEvents as json objects to the broker URL)

Sources: [20]

7.7 Task 6 - Knative Eventing

**Create a target service for CloudEvents,
which upon each request replies with another CloudEvent**

replyfunc/app.py:

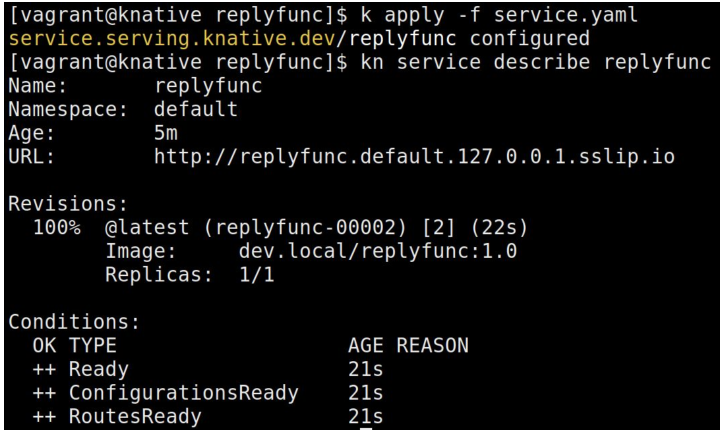


```python
import os, json, uuid


from flask import Flask, make_response, request
import time

app = Flask(__name__)
app.debug = True

@app.route('/', methods = ['POST'])
def hello_world():
    time.sleep(1)
    app.logger.debug(str(request.__dict__))
    # Respond with another event
    response = make_response({
        "msg": "replyfunc"
    })
    response.headers["Ce-Id"] = str(uuid.uuid4())
    response.headers["Ce-specversion"] = "1.0"
    response.headers["Ce-Source"] = "the/replyfunc"
    response.headers["Ce-Type"] = "replytype"
    return response

if __name__ == "__main__":
    app.run(debug=True,host='0.0.0.0',port=int(os.environ.get('PORT', 8080)))
```

Sources: [19]

7.7 Task 6 - Knative Eventing

```
Create a target service for the CloudEvents,
which upon each request replies with another CloudEvent

$ cd replyfunc
$ docker build -t replyfunc:1.0 .
$ docker tag replyfunc:1.0 dev.local/replyfunc:1.0
$ kind load docker-image dev.local/replyfunc:1.0 -n knative
$ k apply -f service.yaml
$ kn service describe replyfunc
```

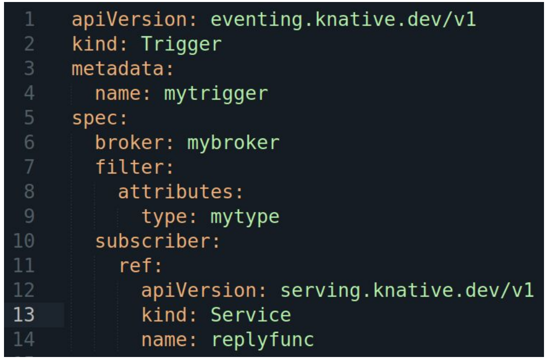


```
[vagrant@knative replyfunc]$ k apply -f service.yaml
service.serving.knative.dev/replyfunc configured
[vagrant@knative replyfunc]$ kn service describe replyfunc
Name:        replyfunc
Namespace:   default
Age:         5m
URL:         http://replyfunc.default.127.0.0.1.sslip.io

Revisions:
  100%  @latest (replyfunc-00002) [2] (22s)
        Image:     dev.local/replyfunc:1.0
        Replicas:  1/1

Conditions:
  OK TYPE                    AGE REASON
  ++ Ready                   21s
  ++ ConfigurationsReady     21s
  ++ RoutesReady             21s
```

7.7 Task 6 - Knative Eventing

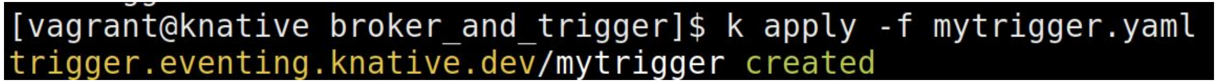
b) Create a trigger listening for events on the new broker, which forwards CloudEvents to one of your already created knative services. The trigger shall only react to CloudEvents having a type with a value of your choice

mytrigger.yaml:



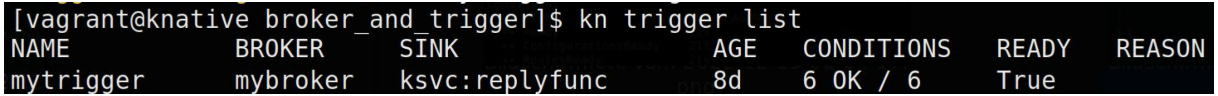
```yaml
apiVersion: eventing.knative.dev/v1
kind: Trigger
metadata:
  name: mytrigger
spec:
  broker: mybroker
  filter:
    attributes:
      type: mytype
  subscriber:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: replyfunc
```

$ k apply -f mytrigger.yaml



```
[vagrant@knative broker_and_trigger]$ k apply -f mytrigger.yaml
trigger.eventing.knative.dev/mytrigger created
```

$ kn trigger list

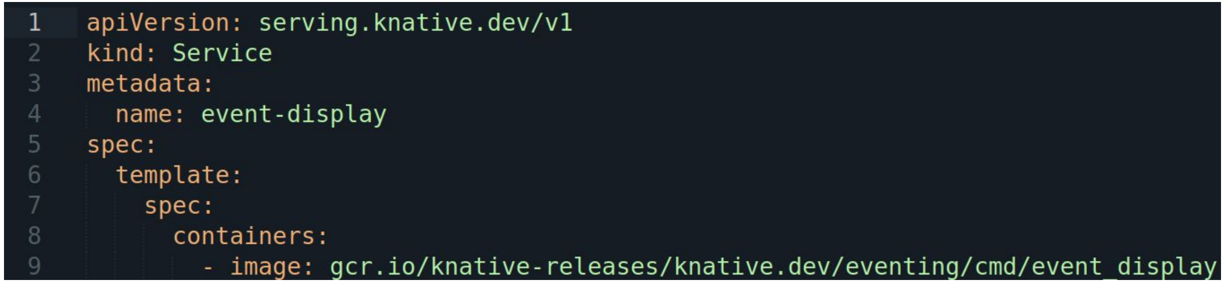


```
[vagrant@knative broker_and_trigger]$ kn trigger list
NAME           BROKER     SINK            AGE   CONDITIONS   READY   REASON
mytrigger      mybroker   ksvc:replyfunc  8d    6 OK / 6     True
```

7.7 Task 6 - Knative Eventing

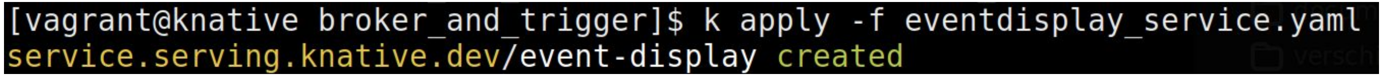
Creating an event-display service, which will dump all incoming CloudEvents to stdout

eventdisplay_service.yaml:



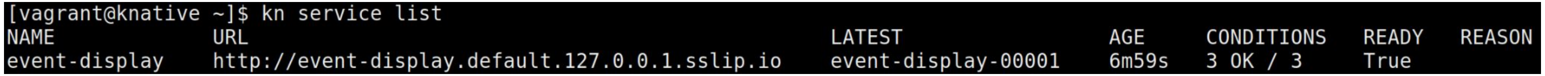
```yaml
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: event-display
spec:
  template:
    spec:
      containers:
        - image: gcr.io/knative-releases/knative.dev/eventing/cmd/event_display
```

$ k apply -f eventdisplay_service.yaml



```
[vagrant@knative broker_and_trigger]$ k apply -f eventdisplay_service.yaml
service.serving.knative.dev/event-display created
```

$ kn service list

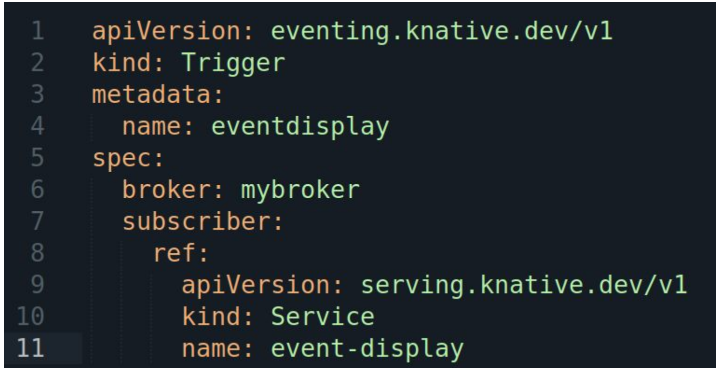


```
[vagrant@knative ~]$ kn service list
NAME            URL                                           LATEST                AGE     CONDITIONS   READY   REASON
event-display   http://event-display.default.127.0.0.1.sslip.io   event-display-00001   6m59s   3 OK / 3     True
```

7.7 Task 6 - Knative Eventing

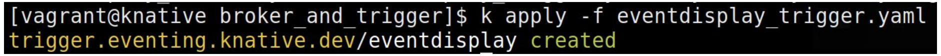
Creating a trigger, which will forward each incoming CloudEvent to the event-display service

eventdisplay_trigger.yaml:



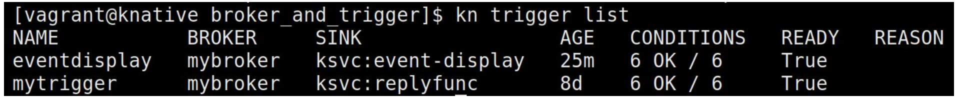
```yaml
apiVersion: eventing.knative.dev/v1
kind: Trigger
metadata:
  name: eventdisplay
spec:
  broker: mybroker
  subscriber:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
```

$ k apply -f eventdisplay_trigger.yaml



```
[vagrant@knative broker_and_trigger]$ k apply -f eventdisplay_trigger.yaml
trigger.eventing.knative.dev/eventdisplay created
```

$ kn trigger list



```
[vagrant@knative broker_and_trigger]$ kn trigger list
NAME           BROKER     SINK                 AGE   CONDITIONS   READY   REASON
eventdisplay   mybroker   ksvc:event-display   25m   6 OK / 6     True
mytrigger      mybroker   ksvc:replyfunc       8d    6 OK / 6     True
```

7.7 Task 6 - Knative Eventing

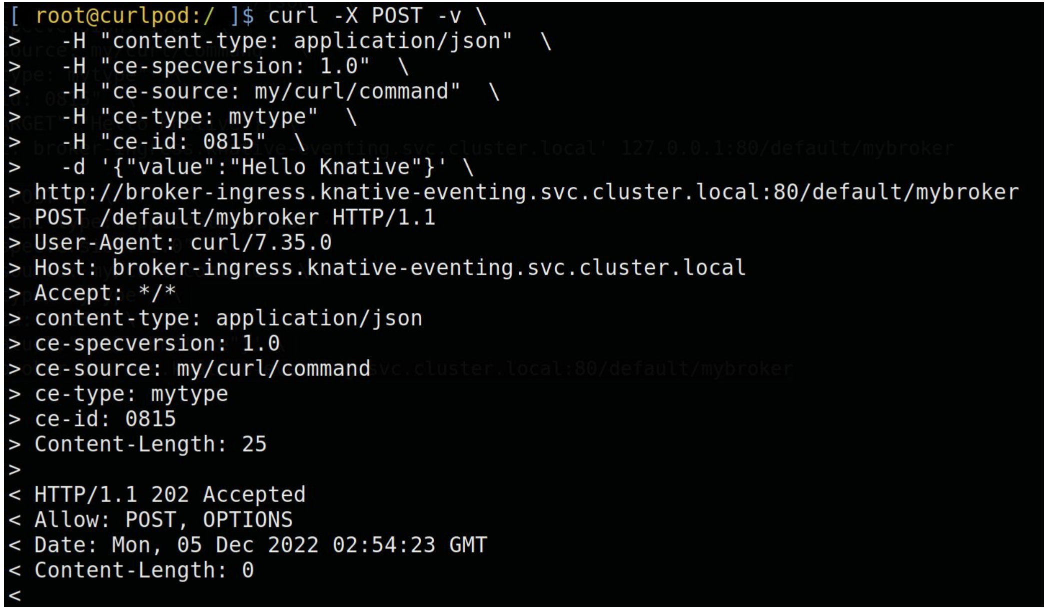
c) Verify the correct functionality of the trigger by sending a CloudEvent to your broker
(You need to send the event from a pod inside the kubernetes cluster to reach the broker url,
or you will get http status code 404 from the ingress controller,
I recommend using curl to send a simple event from a pod within the cluster)

```
$ kubectl -n default run curlpod --image=radial/busyboxplus:curl -i --tty

$ curl -X POST -v \
  -H "content-type: application/json"  \
  -H "ce-specversion: 1.0"  \
  -H "ce-source: my/curl/command"  \
  -H "ce-type: mytype"  \
  -H "ce-id: 0815"  \
  -d '{"value":"Hello Knative"}' \
http://broker-ingress.knative-eventing.svc.cluster.local:80/default/mybroker
```

Sources: [18]

7.7 Task 6 - Knative Eventing

c) Verify the correct functionality of the trigger by sending a cloud-event to your broker
(You need to send the event from a pod inside the kubernetes cluster to reach the broker url,
or you will get http status code 404 from the ingress controller,
I recommend using curl to send a simple event from a pod within the cluster)



```
[ root@curlpod:/ ]$ curl -X POST -v \
>   -H "content-type: application/json"  \
>   -H "ce-specversion: 1.0"  \
>   -H "ce-source: my/curl/command"  \
>   -H "ce-type: mytype"  \
>   -H "ce-id: 0815"  \
>   -d '{"value":"Hello Knative"}' \
> http://broker-ingress.knative-eventing.svc.cluster.local:80/default/mybroker
> POST /default/mybroker HTTP/1.1
> User-Agent: curl/7.35.0
> Host: broker-ingress.knative-eventing.svc.cluster.local
> Accept: */*
> content-type: application/json
> ce-specversion: 1.0
> ce-source: my/curl/command
> ce-type: mytype
> ce-id: 0815
> Content-Length: 25
>
< HTTP/1.1 202 Accepted
< Allow: POST, OPTIONS
< Date: Mon, 05 Dec 2022 02:54:23 GMT
< Content-Length: 0
<
```

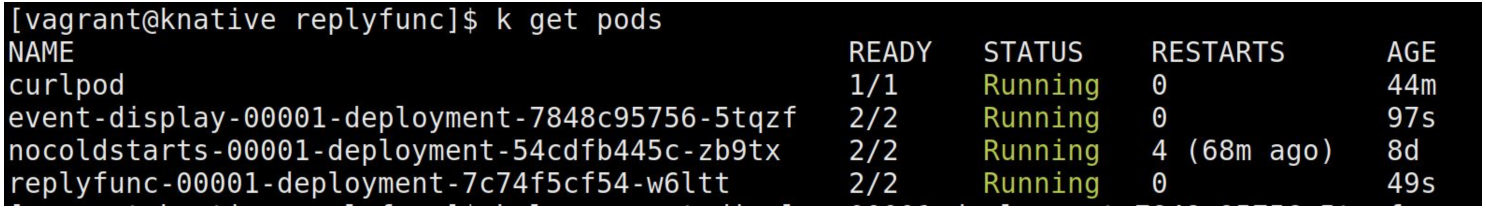
(Correct submission is indicated by http status code 202)

Sources: [18]

7.7 Task 6 - Knative Eventing

c) Verify the correct functionality of the trigger by sending a CloudEvent to your broker (You need to send the event from a pod inside the kubernetes cluster to reach the broker url, or you will get http status code 404 from the ingress controller, I recommend using curl to send a simple event from a pod within the cluster)

```
$ k get pods
```

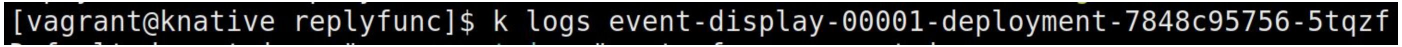


```
[vagrant@knative replyfunc]$ k get pods
NAME                                              READY   STATUS    RESTARTS      AGE
curlpod                                           1/1     Running   0             44m
event-display-00001-deployment-7848c95756-5tqzf   2/2     Running   0             97s
nocoldstarts-00001-deployment-54cdfb445c-zb9tx    2/2     Running   4 (68m ago)   8d
replyfunc-00001-deployment-7c74f5cf54-w6ltt       2/2     Running   0             49s
```

7.7 Task 6 - Knative Eventing

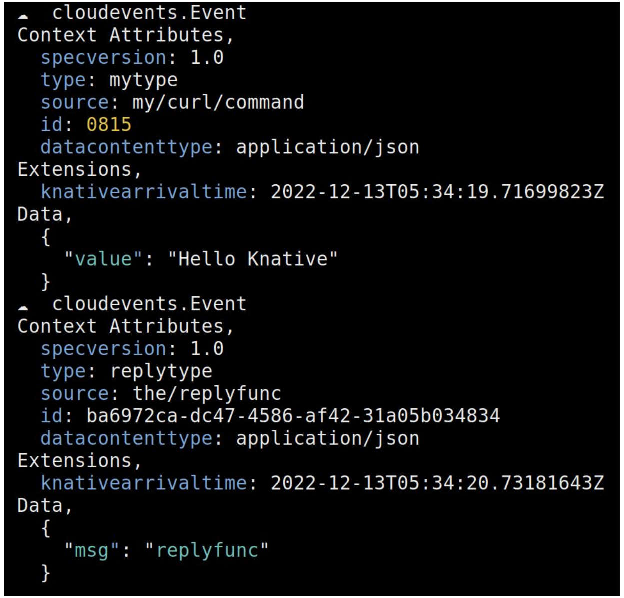
c) Verify the correct functionality of the trigger by sending a CloudEvent to your broker
(You need to send the event from a pod inside the kubernetes cluster to reach the broker url,
or you will get http status code 404 from the ingress controller,
I recommend using curl to send a simple event from a pod within the cluster)

```
$ k logs event-display-00001-deployment-7848c95756-5tqzf
```

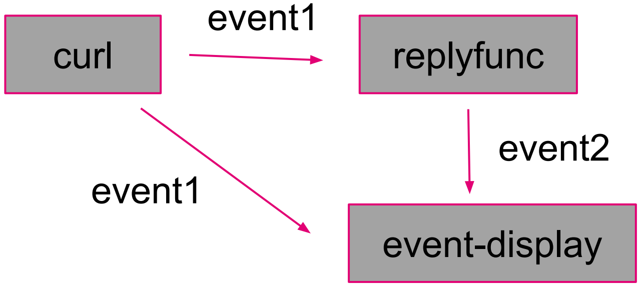


```
[vagrant@knative replyfunc]$ k logs event-display-00001-deployment-7848c95756-5tqzf
```

```
[...]
```



```
☁️  cloudevents.Event
Context Attributes,
  specversion: 1.0
  type: mytype
  source: my/curl/command
  id: 0815
  datacontenttype: application/json
Extensions,
  knativearrivaltime: 2022-12-13T05:34:19.71699823Z
Data,
  {
    "value": "Hello Knative"
  }
☁️  cloudevents.Event
Context Attributes,
  specversion: 1.0
  type: replytype
  source: the/replyfunc
  id: ba6972ca-dc47-4586-af42-31a05b034834
  datacontenttype: application/json
Extensions,
  knativearrivaltime: 2022-12-13T05:34:20.73181643Z
Data,
  {
    "msg": "replyfunc"
  }
```



End of presentation

Thank you for your participation

Feel free to ask questions

Sources (1 / 3)

01. Knative (Hrsg.). (kein Datum). Abgerufen am 05. Dezember 2022 von https://github.com/knative/serving/blob/main/config/core/configmaps/autoscaler.yaml

02. Knative (Hrsg.). (kein Datum). Abgerufen am 05. Dezember 2022 von https://knative.dev/docs/serving/autoscaling/scale-to-zero/

03. Knative (Hrsg.). (kein Datum). Abgerufen am 05. Dezember 2022 von https://knative.dev/docs/serving/autoscaling/scale-bounds/

04. Knative (Hrsg.). (kein Datum). Abgerufen am 05. Dezember 2022 von https://knative.dev/docs/serving/autoscaling/concurrency/

05. Knative (Hrsg.). (kein Datum). Abgerufen am 05. Dezember 2022 von https://github.com/knative/serving/blob/main/config/core/configmaps/defaults.yaml

06. Knative (Hrsg.). (kein Datum). Abgerufen am 05. Dezember 2022 von https://knative.dev/docs/serving/load-balancing/target-burst-capacity/

07. Knative (Hrsg.). (kein Datum). Abgerufen am 05. Dezember 2022 von https://knative.dev/docs/concepts/serving-resources/revisions/

08. Knative (Hrsg.). (kein Datum). Abgerufen am 05. Dezember 2022 von https://knative.dev/docs/serving/traffic-management/

Sources (2 / 3)

09. Knative (Hrsg.). (kein Datum). Abgerufen am 05. Dezember 2022
von https://knative.dev/docs/serving/autoscaling/#additional-resources

10. Knative (Hrsg.). (kein Datum). Abgerufen am 05. Dezember 2022
von https://knative.dev/docs/serving/using-a-custom-domain/

11. Knative (Hrsg.). (kein Datum). Abgerufen am 05. Dezember 2022
von https://knative.dev/docs/serving/services/custom-domains/

12. Knative (Hrsg.). (kein Datum). Abgerufen am 05. Dezember 2022
von https://knative.dev/docs/eventing/brokers/

13. Mete Atamel. (12. März 2020). Abgerufen am 05. Dezember 2022
von https://medium.com/google-cloud/knative-eventing-delivery-methods-79d4ebe30a68

14. Knative (Hrsg.). (kein Datum). Abgerufen am 05. Dezember 2022
von https://github.com/knative/specs/blob/main/specs/eventing/overview.md

15. Knative (Hrsg.). (kein Datum). Abgerufen am 05. Dezember 2022
von https://knative.dev/docs/eventing/triggers/

16. Knative (Hrsg.). (kein Datum). Abgerufen am 05. Dezember 2022
von https://knative.dev/docs/eventing/channels/

Sources (3 / 3)

17. Knative (Hrsg.). (kein Datum). Abgerufen am 05. Dezember 2022
von https://knative.dev/docs/eventing/channels/subscriptions/

18. Knative (Hrsg.). (kein Datum). Abgerufen am 05. Dezember 2022
von https://knative.dev/docs/eventing/brokers/broker-admin-config-options/#procedure

19. Knative (Hrsg.). (kein Datum). Abgerufen am 05. Dezember 2022
von https://github.com/knative/docs/tree/main/code-samples/eventing/helloworld/helloworld-python

20. Knative (Hrsg.). (kein Datum). Abgerufen am 05. Dezember 2022
von https://knative.dev/docs/eventing/brokers/create-mtbroker/

21. Knative (Hrsg.). (kein Datum). Abgerufen am 05. Dezember 2022
von https://knative.dev/docs/eventing/flows/sequence/sequence-reply-to-event-display/#create-the-sequence